robotron

Systemunterlagendokumentation

MOS K1520

Anwenderdokumentation

Anleitung für den Programmierer Teil II

unter dem Betriebssystem SCPX 1526

Systemunterlagen-dokumentation

Anwenderdokumentation

MOS K1520

Stand: 30.4.84

Anleitung fuer den Programmierer Teil II

VEB Robotron Buchungsmaschinenwerk
Karl-Marx-Stadt 1984

Die vorliegende 1. Auflage der - Dokumentation "Anleitung fuer den Programmierer" - entspricht dem Stand April 1984 und unterliegt nicht dem Aenderungsdienst.

Nachdruck, jegliche Vervielfaeltigung oder Auszuege daraus sind unzulaessig.

Die Dokumentation wurde durch ein Kollektiv des VEB Robotron Buchungsmaschinenwerk Karl-Marx-Stadt ausgearbeitet.
Im Interesse einer staendigen Weiterentwicklung werden die Leser gebeten, dem Herausgeber ihre Vorschlaege bzw. Hinweise zur Verbesserung mitzuteilen.

Anmerkung:
Die Anwenderdokumentation zum SCP - System besteht per 7/84 aus bzw. ist zu den angegebenen Terminen vorgesehen:

- Anleitung fuer den Bediener SCP 1520	7/84
- Anleitung fuer den Programmierer SCP 1520 Teil I und Teil II (Sprachbeschreibung ASM)	7/84
- Anleitung fuer den Systemprogrammierer SCP 1520	7/84
- Hardwarebeschreibung	7/84
- Anwenderdokumentation BASIC- Interpreter	7/84
- Anwenderdokumentation BASIC- Compiler	7/84
- Anwenderdokumentation C- Compiler	9/84
- Anwenderdokumentation PASCAL	3/85
- Anwenderdokumentation FORTRAN	3/85
- Anwenderdokumentation Textverarbeitungssystem TP	7/84
- Anwenderdokumentation Installierungs- Programm fuer TP	7/84
- Anwenderdokumentation KOMBO- Druck	10/84
- Schulungshandbuch fuer das Textverarbeitungs- system TP	12/84

VEB Robotron-Buchungsmaschinenwerk
Karl-Marx-Stadt
Software-Zentrum

9010 Karl-Marx-Stadt

Postschliessfach 129

Inhaltsverzeichnis

Seite

Teil I

1. Einleitung

2. Grundlagen fuer die Programmierung
2.1. Speicherkonzept
2.2. Logische Geraete und ihr Aufruf
2.3. Kommandoeingabe

3. BDOS- Funktionen
3.1. Vorbemerkungen
3.2. BDOS- Funktion Ø
3.3. BDOS- Funktionen fuer Konsole und Drucker
3.4. BDOS- Funktionen fuer den Diskettenzugriff

4. Beispiel fuer die Anwendung der BDOS- Funktionen

5. EDIT 1520 (SCPX)

6. LINK 1520 (SCPX)

7. LIB 1520 (SCPX)

8. DU 1520 (SCPX)

Anlage A Steuerzeichen fuer Bildschirm

Anlage B Steuerzeichen fuer Drucker

Anlage C Codetabelle ASCII-Zeichen

Anlage D Uebersicht der Funktionstastencodes und deren Wirkung fuer die Tastaturen K 7637, K 7636/7634 und K 7606/7604 (unter CCP-Regie)

Anlage E Belegte Toradressen und deren Bedeutung

Teil II

9. ASM 1520 (SCPX)

9. ASM 1520 (SCPX)

Inhalt

		Seite
9.1.	Einleitung	6
9.2.	Ueberblick	6
9.3.	Quelldateiorganisation	7
9.3.1.	Formate	7
9.3.1.1.	Quellzeilen-Format	7
9.3.1.2.	Kommentar	8
9.3.2.	Symbole	8
9.3.2.1.	Label (Marke)	9
9.3.2.2.	Public (Eintrittspunkt)	9
9.3.2.3.	External (Externes Symbol)	10
9.3.2.4.	Speicherzuordnungszaehler-Modus	11
9.3.3.	Operationscodes und Pseudooperationen	13
9.3.4.	Argumente (Ausdruecke)	14
9.3.4.1.	Operanden	14
9.3.4.2.	Operatoren	18
9.4.	Assembler-Eigenschaften	22
9.4.1.	Einzelfunktions-Pseudooperationen	22
9.4.1.1.	Pseudooperationen zur Auswahl der Anweisungsliste	23
9.4.1.2.	Pseudooperationen zur Datei- und Symboldefinition	23
9.4.1.3.	Pseudooperation zur Zuweisung des Speicherplatzzuordnungszaehler-Modus	28
9.4.1.4.	Dateibezogene Pseudooperationen	33
9.4.1.5.	Pseudooperationen zur Listensteuerung	37
9.4.2.	Makrofaehigkeit	42
9.4.2.1.	Pseudooperationen zur Makrodefinition	43
9.4.2.2.	Wiederholungs-Pseudooperationen	45
9.4.2.3.	Beendigungs-Pseudooperationen	48
9.4.2.4.	Pseudooperation fuer Makro-Symbole	49
9.4.2.5	Spezielle Makro-Operatoren	50
9.4.3.	Pseudooperationen zur bedingten Assemblierung	51
9.5.	Abarbeitung des Assemblers	54
9.5.1.	Aufrufen des Assmenblers	55
9.5.2.	Die Kommandozeile des Assemblers	55
9.5.2.1.	Source (=dateiname)	56
9.5.2.2.	Object (dateiname)	57
9.5.2.3.	List (,dateiname)	57
9.5.2.4.	Schalter (/switch)	58
9.5.2.5.	Zusaetzliche Angaben der Kommandozeile	61
9.5.3.	Format des Assemblerprotokolls	64
9.5.3.1.	Format der Assemblerliste	64
9.5.3.2.	Format der Symboltabelle	66
9.5.4.	Fehlercodes und Fehlermeldungen	66
9.5.4.1.	Fehlercodes	67
9.5.4.2.	Meldungen	68

Inhalt Seite

9.6.	Beschreibung der Befehle	69
9.6.1.	Ladebefehle	69
9.6.1.1.	8-Bit-Ladebefehle	70
9.6.1.2.	16-Bit-Ladebefehle	72
9.6.2.	Indirekte Registeroperationen	75
9.6.2.1.	PUSH-Befehle	75
9.6.2.2.	POP-Befehle	75
9.6.3.	Register-Austausch-Befehle	77
9.6.4.	Blocktransportbefehle	78
9.6.5.	Blocksuchbefehle	79
9.6.6.	Arithmetische und logische Operationen	80
9.6.6.1.	8-Bit-Arithmetik	81
9.6.6.2.	16-Bit-Arithmetik	83
9.6.6.3.	8-Bit-Logikbefehle	84
9.6.7.	Sprungbefehle	85
9.6.8.	Verschiebebefehle	87
9.6.9.	Spezielle Akkumulator- und Flagbefehle	90
9.6.10.	Unterprogramm-Aufruf-Befehle	91
9.6.11.	Unterprogramm-Ruecksprung-Befehle	92
9.6.12.	CPU-Steuerbefehle	94
9.6.13.	Bittest- und -Setzbefehle	95
9.6.14.	Eingabebefehle	96
9.6.15.	Ausgabebefehle	98
9.6.16.	Abkuerzungsverzeichnis zur Befehlsbeschreibung	99
9.6.17.	Arbeit mit den Bedingungsbits	100
9.7.	Cross-Referenz	103
9.7.1.	Erzeugen einer Cross-Referenz-Liste	103
9.7.1.1.	Erzeugen einer Cross-Referenz-Datei	103
9.7.1.2.	Generieren einer Cross-Referenz-Liste	104
9.7.2.	Pseudooperationen zur Listensteuerung	105
9.8.	Uebersicht ueber die Pseudo-operationen des Assemblers	106
9.9.	Mikrobefehlsliste - Z80-Operationscodes	109
9.10.	Mikrobefehlsliste - 8080-Operations-codes	138

9.1. Einleitung

Der Assembler benoetigt 19 K Speicherplatz plus 4 K Pufferbereich.

Syntax-Notation

[ ]	Die Eingabe ist optional (wahlfrei).
< >	Ein Text in Kleinbuchstaben in spitzen Klammern muss durch den Anwender bereitgestellt werden (Bsp. <dateiname>). Ein Text in Grossbuchstaben gibt eine Tastenbedienung an (Bsp. <ENTER>).
{ }	Auswahl zwischen 2 oder mehreren Eintragungen
...	Die Eintragung kann wiederholt werden.

Alle anderen Interpunktionszeichen

, : = / * ¤ .

werden unveraendert uebernommen.

9.2. Ueberblick

a) Der Assembler unterstuetzt 2 Assemblersprachen:

- 8080-Mnemonik
- Z80-Mnemonik

b) Der Assembler erzeugt relativen und/oder absoluten Code.

c) Der Assembler unterstuetzt das Schreiben von Makros.
Der Programmierer schreibt einen Block fuer eine Anzahl Anweisungen. Dieser Block erhaelt einen Namen, mit dem der Makro aufgerufen wird. Die Anweisungen sind die Makro-Definition.
Wenn diese Befehle an irgendeiner Stelle benoetigt werden, ruft der Programmierer den Makro auf. Der Makro-Aufruf uebergibt ausserdem die Parameter an den Assembler fuer die Makro-Expansion.
Das Verwenden von Makros reduziert den Umfang fuer eine Quelle, da die Makro-Definitionen in einer gesonderten Datei auf der Diskette stehen koennen und nur in den Modul uebernommen werden, wenn sie waehrend der Assemblierung gefordert werden.
Makros koennen geschachtelt werden, d.h. in einem Makro kann ein anderer aufgerufen werden. Die Schachtelungstiefe ist nur durch den Speicherplatz begrenzt.

d) **bedingte Assemblierung**
Der Assembler unterstuetzt die bedingte Assemblierung. Der Programmierer kann eine Bedingung bestimmen, unter welcher Teile des Programms entweder assembliert oder nicht assembliert werden.
Die Moeglichkeiten der bedingten Assemblierung werden durch einen kompletten Satz bedingter Pseudooperationen vergroessert, welche das Testen der Assembler-Paesse, der Symbol-Definitionen und der Makro-Parameter einschliessen. Die Bedingungen koennen bis zu 255-mal geschachtelt werden.

9.3. Quelldatei-Organisation

9.3.1. Formate

Eine Assembler-Quelldatei ist eine Anzahl von Zeilen geschrieben in Assembler-Sprache.
Die letzte Zeile der Datei muss eine END-Anweisung sein.
Entsprechende Anweisungen, wie z.B. IF...ENDIF, muessen in der richtigen Reihenfolge stehen. Anderenfalls koennen die Zeilen in beliebiger Reihenfolge des Programmentwurfs erscheinen.

9.3.1.1. Quellzeilen-Format

Eine Eingabe-Quelldatei fuer den Assembler besteht aus Anweisungs-Zeilen, die in Teile oder Felder unterteilt sind:

Symbol Operation Argument Kommentar

Symbol: Dieses Feld kann eines der drei Symboltypen enthalten:
. Label (Marke)
. Public
. External,
gefolgt von einem Doppelpunkt, ausser wenn das Symbol Teil einer SET- ,EQU- oder MAKRO-Anweisung ist.

Operation: Dieses Feld enthaelt einen Operationscode, eine Pseudooperation, einen Makro-Namen oder einen Ausdruck.

Argument: Dieses Feld enthaelt Ausdruecke, Variable, Register-Namen, Operanden und Operatoren.

Kommentar: Dieses Feld enthaelt Kommentartext, der immer mit einem Semikolon beginnen muss.

Alle Felder sind optional. Auch Leerzeilen sind moeglich. Die Anweisungen beginnen in einer beliebigen Spalte. Mehrere Leerzeichen oder Tabulatoren zwischen den Feldern koennen zur besseren Lesbarkeit eingefuegt werden, aber mindestens ein Leerzeichen oder Tabulator ist zwischen zwei Feldern noetig.

9.3.1.2. Kommentar

Ein Kommentar beginnt immer mit einem Semikolon und endet mit <ENTER>.
Fuer lange Kommentare kann man die .COMMENT-Pseudooperation verwenden, um das Einfuegen des Semikolons auf jeder Zeile zu vermeiden.

9.3.2. Symbole

Symbole sind Namen fuer Teilfunktionen oder Werte. Symbolnamen werden durch den Programmierer definiert. Die Symbole fuer dieses Programmpaket gehoeren zu einem von drei Typen, gemaess ihrer Funktion.
Diese 3 Typen sind:

- Label
- Public
- External

Alle 3 Typen haben ein Modus-Attribut, das mit dem Speichersegment korrespondiert, welches das Symbol repraesentiert (siehe Pkt. 9.3.2.4.).

Alle 3 Typen der Symbole haben folgende Charakteristik:

1. Symbole koennen beliebige Laenge haben, aber die Anzahl der signifikanten Zeichen, die an den Programmverbinder uebergeben werden, varriiert mit dem Typ des Symbols:

   a) Fuer Labels (Marken) sind die ersten 16 signifikant.
   b) Fuer Public- und External-Symbole werden die ersten 6 Zeichen an den Programmverbinder uebergeben.

   Zusaetzliche Zeichen werden "intern" abgeschnitten.

2. Ein gueltiges Symbol kann folgende Zeichen enthalten:
   A-Z 0-9 ¤ . ? @ _

3. Ein Symbol darf nicht mit einer Ziffer oder einem Unterstreichungszeichen beginnen.

4. Symbole, die mit gueltigen Registernamen oder Bedienungscodes uebereinstimmen, sind nicht erlaubt.

5. Beim Lesen eines Symbols werden Kleinbuchstaben in Grossbuchstaben gewandelt. Es koennen also auch Klein- und Grossbuchstaben in einem Symbol gemischt auftreten.

9.3.2.1. Label (Marke)

Ein Label ist ein Referenzpunkt fuer eine Anweisung **innerhalb** des Programmoduls, in dem diese Marke definiert ist.
Eine Marke setzt ihren Adresswert auf die Adresse der Daten, die der Marke zugeordnet sind.

Beispiel:

```
BUFF: DS 1ØØØH
```

BUFF ist gleich der 1. Adresse der 1ØØØH Bytes des reservierten Platzes.

Ist eine Marke einmal definiert, kann sie als Eintragung im Argumentfeld verwendet werden.
Eine Anweisung mit einer Marke in ihrem Argument bezieht sich auf die Anweisungszeile, in welcher diese Marke im Symbolfeld als Marke definiert ist.

Beispiel:

```
STA   BUFF
```

schreibt den Inhalt des A-Registers auf den Speicherplatz, den die Marke definiert.

Eine Marke kann ein beliebiges gueltiges Symbol mit bis zu 16 Zeichen Laenge sein.
Wenn eine Marke definiert werden soll, so muss sie als erstes in der Anweisungszeile stehen.
Sowohl bei Z80- als auch bei 8080-Mnemonik muss nach einer Marke sofort **ein** Doppelpunkt (kein Leerzeichen) folgen; es sei denn, die Marke ist Teil einer SET- oder EQU-Anweisung (dann kein Doppelpunkt).
(Folgen der Marke 2 Doppelpunkte, dann wird sie zum Public-Symbol!)

9.3.2.2. Public

Ein Public-Symbol ist als Marke definiert. Der Unterschied zur Marke besteht darin, dass ein Public-Symbol auch als Referenzpunkt **in anderen Programmoduln** verfuegbar ist.

Es gibt 2 Moeglichkeiten, ein Symbol als Public zu erklaeren:

1. 2 Doppelpunkte (::) folgen dem Namen

```
MARKE:: RET
```

2. Verwendung einer der Pseudo-Operationen
   PUBLIC, ENTRY oder GLOBAL

```
        PUBLIC MARKE
```

Beispiel:

Das Resultat beider Moeglichkeiten ist identisch:

```
        MARKE:: RET
```

ist aequvalent zur Anweisungsfolge

```
        PUBLIC  MARKE
        MARKE:  RET
```

9.3.2.3. External

Ein externes Symbol ist ausserhalb des aktuellen Programmoduls in einem anderen, separaten Modul als Public-Symbol definiert.

Die Wertzuweisung erfolgt waehrend der Programmverbindung. Dabei wird dem externen Symbol der Wert des Public-Symbols aus dem anderen Programmodul uebergeben.

Beispiel:

```
    MODUL1

    MARKE::     DB          7           ;PUBLIC  MARKE=7

    MODUL2

                BYTE EXT    MARKE       ;EXTERNAL MARKE
```

Waehrend der Programmverbindung wird fuer EXTERNAL MARKE der Wert von PUBLIC MARKE verwendet.

Ein Symbol wird zum External erklaert durch:

1. 2 Doppelkreuze (##) nach dem Namen des Symbols.

   Beispiel:

   ```
        CALL  MARKE##
   ```

   erklaert MARKE als ein Symbol mit einem 2-Byte-Adresswert, definiert in einem anderen Modul.

2. Fuer 2-Byte-Werte eine der Pseudo-Operationen

   EXT, EXTRN oder EXTERNAL

   Beispiel:

   EXT MARKE

   erklaert MARKE als ein 2-Byte-Symbol, definiert in einem anderen Modul.

3. Fuer 1-Byte-Werte eine der Pseudooperationen

   BYTE EXT ,BYTE EXTRN oder BYTE EXTERNAL

   Beispiel:

   BYTE EXT MARKE

   erklaert MARKE als 1-Byte-Wert, definiert in einem anderen Programmodul.

Das Ergebnis aller Moeglichkeiten der Vereinbarung ist dasselbe.

```
CALL  MARKE##
```

ist aequivalent zur Anweisungsfolge

```
EXT   MARKE
CALL  MARKE
```

9.3.2.4. Speicherzuordnungszaehler-Modus (Typenarten)

Ein Symbol bezieht sich beim Auftreten seines Namens im Argumentfeld auf eine Anweisungszeile.
Der Wert des Symbols ist die Adresse der Anweisung, die durch das Symbol markiert wird. Wenn das Symbol im Argumentfeld auftritt, wird es druch seinen Wert ersetzt und in der Operation verwendet.

Der Wert eines Symbols wird gemaess seinem Speicherzuordnungszaehler (PC)-Modus bestimmt.

Durch den Speicherzuordnungszaehler-Modus ist eine Teilung des Speichers in 4 Segmente moeglich. Er bestimmt, in welches Segment der Programmteil geladen wird.

Die 4 Segmente sind:

- absolutes Segment
- coderelatives Segment
- datenrelatives Segment
- commonrelatives Segment

Der PC-Modus legt fest, ob ein Programmteil

- in den Speicher auf eine absolute, vom Programmierer festgelegte Adresse, geladen wird (absoluter Modus)

- oder auf relative Adressen, die sich aendern, abhaengig von der Groesse und der Anzahl der Programme (coderelativer Modus) und dem Umfang der Daten (datenrelativer Modus)

- oder auf Adressen, die gemeinsame Bereiche mit anderen Programmen darstellen (commonrelativer Modus).

Wurde nichts angegeben, ist der Modus coderelativ (standard). Ein Programm kann Programmteile fuer verschiedene Speichersegmente enthalten.

Absoluter Modus

Der absolute Modus erzeugt nichtverschiebbaren Code. Der Programmierer waehlt den absoluten Modus, wenn ein Programmblock stets auf festgelegte absolute Adressen zu laden ist, unabhaengig von sonstigen zu ladenden Segmenttypen (DSEG, CSEG, COMMON).

Datenrelativer Modus

Der datenrelative Modus erzeugt verschieblichen Code fuer einen Programmteil, der veraenderlichen Code enthalten kann und folglich in den RAM-Bereich geladen werden muss. Dies bezieht sich besonders auf Programmdatenbereiche. Symbole im datenrelativen Modus sind verschieblich.

Coderelativer Modus

Der Code- (programm-) relative Modus erzeugt Code fuer einen verschieblichen Programmteil, dessen Codeinhalt sich nicht veraendert. Programmteile fuer PROM-Abspeicherung muessen im coderelativen Modus erzeugt werden.

Common-relativer Modus

Der Common-relative Modus erzeugt Code, der in einem definierten Common-Bereich (gemeinsamen Datenbereich) geladen wird. Das ermoeglicht, den Programmodul in einen Speicherblock und gemeinsame Werte zu teilen.

Zum Aendern des Modus wird in einer Anweisungszeile eine PC-Modus-Pseudooperation verwendet.

ASEG	absoluter Modus
DSEG	datenrelativer Modus
CSEG	coderelativer Modus (zugewiesener Modus)
COMMON	commonrelativer Modus

Diese Pseudooperationen sind in Pkt. 9.4.1.3. detailiert beschrieben.
Die PC-Modus-Faehigkeit im Assembler gestattet dem Programmierer, verschiebbare Assemblerprogramme zu entwickeln.

Verschieblich bedeutet, der Programmodul kann auf jede beliebige Adresse auf verfuegbaren Speicher geladen und getestet werden, unter Verwendung des /P- und /D-Schalters (Spezialkommandos) des Programmverbinders.

9.3.3. Operationscodes und Pseudooperationen

Operationscodes sind mnemonische Namen fuer die Maschinenanweisungen (CPU-Befehle).
Pseudooperationen sind Vorschriften fuer den Assembler, nicht fuer den Mikroprozessor.
Operationscodes und Pseudooperationen werden (meist) in das Operationsfeld der Anweisungszeile eingetragen.

Eine Operation kann sein:

- jede beliebige Z80- oder 8080-Mnemonik
- eine Assembler-Pseudooperation
- ein Makro-Aufruf
- ein Ausdruck

Die Eintragungen im Operationsfeld werden vom Assembler in der folgenden Reihenfolge ausgewertet:

1. Makro-Aufruf
2. Operationscode/Pseudooperation
3. Ausdruck

Der Assembler vergleicht die Eintragung im Operationsfeld mit einer internen Liste von Makronamen. Wenn der Name gefunden wurde, wird die Makroexpansion durchgefuehrt und die entstehenden Anweisungen in den Modul eingefuegt (siehe dazu auch Abschnitt 9.4.2.). Ist die Eintragung kein Makro, versucht der Assembler die Eintragung als Operationscode auszuwerten.

War die Eintragung kein Operationscode, versucht der Assembler die Eintragung als Pseudooperation auszuwerten.
Wenn die Eintragung auch keine Pseudooperation ist, wertet der Assembler die Eintragung als Ausdruck

Wenn ein Ausdruck als Anweisungszeile ohne vorangestelltem Operationscode, Pseudooperation oder Makroname geschrieben wurde, meldet der Assembler keinen Fehler. Vielmehr setzt er voraus, dass ein definiertes Byte (Pseudooperation) vor den Ausdruck gehoert und uebersetzt diese Zeile.

Wegen der Auswertungsreihenfolge verhindert ein Makroname, der identisch einem Operationscode ist, desses Verwendung als Operationscode. Dieser Name wird dann ausschliesslich als Makroaufruf gebraucht. Wurde z.B. einem Makro der Name ADD zugewiesen, kann in diesem Programm ADD nicht mehr als Operationscode verwendet werden!

9.3.4. Argumente (Ausdruecke)

Die Argumente fuer die Operationscodes und Pseudooperationen werden gewoehnlich als Ausdruecke bezeichnet, da sie mathematischen Ausdruecken aehneln,

wie z.B. 5+4*3.

Die Teile eines Ausdruckes werden Operanden (5,4 und 3 in diesem mathematischen Ausdruck) und Operatoren (+ und * sind Beispiele) genannt. Ausdruecke koennen einen oder mehrere Operanden beinhalten.

Ein-Operand-Ausdruecke sind die am haeufigsten gebrauchten Argumente.
Wenn ein Ausdruck mehr als einen Operanden hat, sind die Operanden miteinander durch einen Operator verbunden.

Beispiel: 5+4 6-3 7*2 8/7 9>8 usw.

Im Assembler sind die Operanden numerische Werte, vertreten durch Zahlen, Zeichen, Symbole oder 8080-Operationscodes. Die Operatoren koennen arithmetische oder logische sein.

Die folgenden Abschnitte definieren die vom Assembler zugelassenen Operanden und Operatoren.

9.3.4.1. Operanden

Operanden koennen sein

- Zahlen
- Zeichen
- Symbole
- 8080-Operationscodes

Zahlen

Mit diesem Assembler kann in unterschiedlichen Zahlensystemen gearbeitet werden.
Die zugewiesene Basis fuer Zahlen ist dezimal (Standard). Diese Basis kann durch die Pseudooperation .RADIX geaendert werden. Jede Basis von 2 (binaer) bis 16 (hexadezimal) kann ausgewaehlt werden. Wenn die Grundzahl groesser als 1Ø ist, werden fuer die auf 9 folgenden Ziffern A-F verwendet. Wenn das erste Zeichen einer Zahl nicht numerisch ist, muss der Zahl eine Ø vorausgehen.
Eine Zahl wird immer in der augenblicklich zugewiesenen Grundzahl ausgewertet, falls keine der folgenden Bezeichnungen verwendet wurden:

nnnnB	binaer
nnnnD	dezimal
nnnnO	oktal
nnnnH	hexadezimal
Xnnnn	hexadezimal

Zahlen sind vorzeichenlose binaere 16-Bit-Werte (im Wertebereich 0...65535). Der Ueberlauf einer Zahl beim Ueberschreiten von 2 Byte wird ignoriert, das Ergebnis bilden die niederwertigen 16 Bits.

ASCII-Zeichen-Ketten

Eine Zeichenkette wird gebildet aus null oder mehr Zeichen und begrenzt durch Anfuehrungsstriche (") oder Apostroph (').

Wenn eine durch " begrenzte Kette als Argument auftritt, wird der Wert der Zeichen einer nach dem anderen im Speicher abgelegt.

Beispiel: DB "ABC"

speichert den Code von A auf die erste Adresse, B auf Adresse+1 und C auf Adresse+2.

Die Begrenzer koennen als Zeichen verwendet werden, wenn sie fuer jedes gewuenschte Vorkommen zweifach erscheinen.

Beispiel:

```
"Heute ist ein ""schoener"" Tag"
```

speichert die Zeichenkette

Heute ist ein "schoener" Tag.

Stehen keine Zeichen zwischen den Anfuehrungszeichen,wird die Kette als Nullkette ausgewertet (leere Zeichenkette).

Zeichenkonstanten

Diese Ketten, Zeichenkonstanten, setzen sich aus null, einem oder zwei ASCII-Zeichen zusammen, die durch Anfuehrungszeichen (") oder Apostroph (') begrenzt werden. Soll der Begrenzer Bestandteil der Kette sein, so muss er an dieser Stelle doppelt geschrieben werden.

Die Unterschiede zwischen Zeichenkonstanten und Ketten sind:

1. Eine Zeichenkonstante besteht nur aus null, einem oder zwei Zeichen.

2. Die geschriebenen Zeichen sind dann eine Zeichenkonstante, wenn der Ausdruck mehr als einen Operanden hat. Wenn die Zeichen als einziger Operand auftreten, werden sie als Kette ausgewertet und abgespeichert.

   Beispiel:

   ```
   'A'+1 ist eine Zeichenkonstante, aber
   'A' ist eine Zeichenkette.
   ```

3. Der Wert einer Zeichenkonstanten wird berechnet und das Ergebnis wird gespeichert, das niederwertige Byte in die erste Adresse und das hoeherwertige Byte in die zweite Adresse.

   Beispiel:

   ```
   DW 'AB'+Ø
   ```

   wertet 4142H aus und speichert 42H in die erste Adresse und 41H in die zweite Adresse.

Eine Zeichenkonstante, die nur ein Zeichen beinhaltet, bekommt den Code des Zeichens zugeordnet. Das heisst, das hoeherwertige Byte des Wertes ist Null, das niederwertige Byte ist der Code des Zeichens.

Beispiel: Der Wert der Zeichenkonstanten 'A' ist 41ØØH.

Eine Zeichenkonstante, die aus zwei Zeichen besteht, hat als ihren Wert den ASCII-Wert des 1.Zeichens im hoeherwertigen Byte und den ASCII-Wert des 2. Zeichens im niederwertigen Byte.

Zum Beispiel ist der Wert der Zeichenkonstanten

'AB'+Ø gleich 41H*256+42H+Ø.

Die ASCII-Werte dezimal und hexadezimal fuer Zeichen sind im Anhang zusammengefasst.

Symbole in Ausdruecken

Ein Symbol kann als Operand in einem Ausdruck verwendet werden.
Das Symbol wird ausgewertet und der Wert wird fuer das Symbol eingesetzt.

Anwendungsvorschriften fuer das Verwenden externer Symbole in Ausdruecken

1. Externe Symbole koennen in Ausdruecken nur mit folgenden Operatoren verwendet werden:

   + - * / MOD HIGH LOW

2. Wurde in einem Ausdruck ein externes Symbol verwendet, so ist das Ergebnis des Ausdrucks immer extern.

Modusvorschriften fuer Symbole in Ausdruecken

1. Bei allen Operationen, ausser AND, OR und XOR, koennen die Operanden jeden Modus haben.

2. Fuer AND, OR, XOR, SHL und SHR muessen beide Operanden absolut und intern sein.

3. Enthaelt ein Ausdruck einen absoluten Operanden und einen Operanden in einem anderen Modus, wird das Ergebnis immer den anderen, nicht absoluten Modus besitzen.

4. Bei der Subtraktion zweier Operanden verschiedener Modi ist das Ergebnis absolut. Anderenfalls ist das Ergebnis vom Typ der Operanden.

5. Wird ein datenrelatives und ein coderelatives Symbol addiert, ist das Ergebnis unbekannt. Der Assembler uebergibt den Ausdruck an den Programmverbinder als unbekannt und dieser loest den Ausdruck.

Der laufende Speicherzuordnungszaehler

Ein zusaetzliches Symbol fuer das Argumentfeld ist:

. der Speicherzuordnungszaehler.

Der Speicherzuordnungszaehler ist die Adresse der naechsten zu uebersetzenden Anweisung.
Der Speicherzuordnungszaehler ist oft ein passender Bezugspunkt fuer die Berechnung neuer Adressen.

Anstatt sich die laufende Programmadresse zu merken oder zu berechnen, kann der Programmierer ein Symbol verwenden, das dem Assembler den Wert des aktuellen Speicherzuordnungszaehlers mitteilt.

Das laufende Speicherzuordnungszaehlersymbol ist $.

8080-Operationscodes als Operanden

8080-Operationscodes sind nur im 8080-Modus gueltige Ein-Byte-Operanden.

Waehrend der Uebersetzung wird der hexadezimale Wert des Operationscodes berechnet und als Operand verwendet.

Bei der Verwendung von 8080-Operationscodes als Operanden muss zuerst die .8080-Pseudooperation gesetzt werden (siehe Abschnitt 9.4.1.3.).
Nur das erste Byte des errechneten hexadezimalen Wertes ist ein gueltiger Operand.

Die Verwendung von runden Klammern weist den Assembler darauf hin, dass ein Byte fuer den Operationscode zu generieren ist, wo normalerweise mehr als eins generiert wird.

Beispiel:

```
MVI  A,(JMP)
ADI  (CPI)
MVI  B,(RNZ)
CPI  (INX H)
ACI  (LXI B)
```

Der Assembler meldet einen Fehler, wenn der errechnete hexadezimale Wert mehr als 1 Byte (innerhalb der runden Klammern) ergibt, wie bei (CPI 5), (LXI B,LABEL1) oder (JMP LABEL2).
Operationscodes, die normalerweise nur 1 Byte generieren, koennen als Operand ohne einschliessende runde Klammern verwendet werden.

9.3.4.2. Operatoren

Der Assembler erlaubt sowohl arithmetische als auch logische Operatoren.
Operatoren, die wahre oder falsche Bedingungen melden, geben wahr zurueck, wenn das Ergebnis ungleich Ø ist und falsch, wenn das Ergebnis gleich Ø ist.
Die folgenden arithmetischen und logischen Operatoren sind in Ausdruecken erlaubt:

Operator	Definition
NUL	- Gibt wahr zurueck, wenn das Argument (ein Parameter) null ist. Der Rest der Zeile nach NUL wird als Argument zu NUL genommen. Die Bedingung IF NUL <argument> ist falsch, wenn das erste Zeichen des Arguments alles andere als ein Semikolon oder ein <ENTER> ist. Bemerkung: IFB und IFNB fuehren dieselbe Funktion aus, aber sie sind einfacher zu verwenden (siehe Pkt. 9.4.1.5.).
TYPE	- Der TYPE-Operator gibt ein Byte zurueck, das zwei Eigenschaften seines Arguments beschreibt (1) den Modus und (2) ob es extern ist oder nicht. Das Argument von TYPE kann jeder beliebige Ausdruck sein. Wenn der Ausdruck ungueltig ist, gibt TYPE Ø zurueck. Dieses Byte, das von TYPE zurueckgegeben wird, ist wie folgt aufgebaut: bit Ø und 1 geben den Modus an: ØØØØ ØØØØ absolut ØØØØ ØØØ1 coderelativ ØØØØ ØØ1Ø datenrelativ ØØØØ ØØ11 commonrelativ Das 7. Bit ist das externe Bit. Ist das 7. Bit gesetzt, enthaelt der Ausdruck ein EXTERNAL. Besitzt das 7. Bit den Wert Ø, dann ist der Ausdruck lokal (nicht extern). Das Definitionsbit ist das 5. Bit. Dieses Bit ist 1, wenn der Ausdruck lokal definiert wurde und es ist Ø, wenn der Ausdruck undefiniert oder extern ist. TYPE wird meistens innerhalb von Makros verwendet, wo der Modus eines Arguments getestet werden muss, um Entscheidungen betreffs des Programmablaufs zu treffen, z.B. wenn bedingte Assemblierung enthalten ist.

Operator	Definition

Beispiel:

```
MARKE     MACRO    X
          LOCAL    Z
Z         SET TYPE X
IF        Z...
```

TYPE testet den Modus und die Zuordnung von X. Abhaengig von der Auswertung von X wird der Codeblock beginnend mit

IF Z...

uebersetzt oder weggelassen.

LOW - Isolieren der niederwertigen 8 bit (low-Teil) eines absoluten 16-Bit-Wertes.

HIGH - Isolieren der hoeherwertigen 8 bit (high-Teil) eines absoluten 16-Bit-Wertes.

***** - Multiplikation

- Division

MOD - Modulo (Restdivision).
Dividieren des linken Operanden durch den rechten Operanden und Zurueckgeben des Restes als Wert (Modulo).

SHR - Rechtsverschiebung.
SHR gefolgt von einer ganzen Zahl, die die Anzahl der Bit-Positionen angibt, um die der Wert nach rechts verschoben werden soll.

SHL - Linksverschiebung.
SHL gefolgt von einer ganzen Zahl, die die Anzahl der Bit-Positionen angibt, um die der Wert nach links verschoben werden soll.

- (Vorzeichen -Minus) - zeigt an, dass der folgende Wert negativ ist (negative ganze Zahl).

+ - Addition

- - Subtraktion des rechten Operanden vom linken Operanden.

EQ - Gleichheit.
Gibt wahr zurueck, wenn die Operanden einander gleich sind.

Operator	Definition
NE	- Nicht gleich. Gibt wahr zurueck, wenn die Operanden einander nicht gleich sind.
LT	- Kleiner als. Gibt wahr zurueck, wenn der linke Operand kleiner als der rechte Operand ist.
LE	- Kleiner als oder gleich. Gibt wahr zurueck, wenn der linke Operand kleiner oder gleich dem rechten Operanden ist.
GT	- Groesser als. Gibt wahr zurueck, wenn der linke Operand groesser als der rechte ist.
GE	- Groesser als oder gleich. Gibt wahr zurueck, wenn der linke Operand groesser oder gleich dem rechten Operanden ist.
NOT	- Logisches Nicht. Gibt wahr zurueck, wenn der linke Operand wahr und der rechte falsch ist, oder wenn der rechte wahr und der linke falsch ist. Gibt falsch zurueck, wenn beide Operanden wahr oder beide falsch sind.
AND	- Logisches Und. Gibt wahr zurueck, wenn beide Operanden wahr sind; gibt falsch zurueck, wenn einer oder beide Operanden falsch sind. Beide Operanden muessen absolute Werte sein.
OR	- Logisches Oder. Gibt wahr zurueck, wenn einer oder beide Operanden wahr sind. Gibt falsch zurueck, wenn beide Operanden falsch sind. Beide Operanden muessen absolute Werte sein.
XOR	- Exklusives Oder. Gibt wahr zurueck, wenn einer der Operanden wahr und der andere falsch ist. Gibt falsch zurueck, wenn beide Operanden wahr oder beide Operanden falsch sind. Beide Operanden muessen absolute Werte sein.

Die Reihenfolge fuer diese Operatoren ist:

```
NUL,TYPE
LOW,HIGH
*,/,MOD,SHR,SHL
Vorzeichen Minus
+,-
EQ,NE,LT,LE,GT,GE
NOT
AND
OR,XOR
```

Wenn Unterausdruecke Operatoren hoeherer Rangordnung einschliessen, dann wird dieser Ausdruck zuerst berechnet. Die Abarbeitungsreihenfolge kann durch das Verwenden runder Klammern um den Teil des Ausdrucks, der eine hoehere Rangordnung erhalten soll, veraendert werden.

Alle Operatoren ausser +, -, * und / muessen von ihren Operanden durch wenigstens ein Leerzeichen getrennt werden. Die Byte isolierenden Operatoren (HIGH und LOW) trennen die hoeherwertigen oder die niederwertigen 8 Bits eines 16-Bit-Wertes.

9.4. Assembler-Eigenschaften

Der Assembler besitzt 3 generelle Vorzuege:

- Einzelfunktions-Pseudooperationen
- Moeglichkeit der Makroprogrammierung
- bedingte Assemblierung

9.4.1. Einzelfunktions-Pseudooperationen

Einzelfunktions-Pseudooperationen haben nur **eine** Anweisungszeile, sei weisen den Assembler auf die Ausfuehrung nur einer Funktion hin (Makros und Bedingungen haben mehr als eine Code-Zeile; man kann sie als Block von Pseudooperationen auffassen).

Die Einzelfunktions-Pseudooperationen werden in 5 Gruppen eingeteilt:

- Auswahl der Anweisungsliste
- Daten- und Symboldefinition
- Speicherzuordnungszaehler-Modus
- Dateibezogene Pseudooperationen
- Listensteuerung

9.4.1.1. Pseudooperationen zur Auswahl der Anweisungsliste

Standard ist die Z80-Mnemonik.
Wenn nicht die richtige Pseudooperation zur Auswahl der Anweisungsliste angegeben wurde, gibt der Assembler einen schweren Fehler fuer jene Operationscodes zurueck, die fuer die aktuelle Anweisungsliste ungueltig sind. Das heisst, .Z80 uebersetzt nur Z80-Operationscodes und .8080 nur 8080-Operationscodes. Deshalb muss bei einem im 8080-Code geschriebenen Assemblerprogramm die Pseudooperation zur Auswahl dieser Anweisungsliste, .8080 geschrieben werden.

Bemerkung:
Alle in diesem Kapitel aufgelisteten Pseudooperationen werden bei beiden Anweisungslisten verarbeitet.

.Z80

.Z80 hat keine Argumente.
.Z80 weist den Assembler auf die Uebersetzung von Z80-Operationscodes hin.

.8080

.8080 hat keine Argumente.
.8080 weist den Assembler auf die Uebersetzung von 8080- Code hin. Die Auswahl dieses Befehlssatzes ist Standard.

Alle Operationscodes, die der Pseudooperation zur Auswahl des Befehlssatzes folgen, werden entsprechend uebersetzt, bis die alternative Pseudooperation auftritt.
Tritt ein Operationscode auf, der nicht zum ausgewaehlten Befehlssatz gehoert, gibt der Assembler einen "U-Fehler" zurueck.

9.4.1.2. Pseudooperation zur Daten- und Symboldefinition

Alle Programmoperationen zur Definition von Daten und Symbolen werden fuer beide Befehlssaetze unterstuetzt. (Die einzige Ausnahme bildet SET, das fuer die Z80-Anweisungsliste nicht erlaubt ist.)

Definiere Byte

```
DB    <exp>[,<exp>...]
DEFB  <exp>[,<exp ...]
DB    <string>[,<string>...]
DEFM  <string>[,<string>...]
```

Die Argumente der DB's sind entweder Ausdruecke oder Zeichenketten.Die Argumente der DEFB's sind Ausdruecke. Die Argumente der DEFM's sind Ketten. Ketten muessen in einfache oder doppelte Anfuehrungszeichen eingeschlossen werden.

Bemerkung:
DB wird in den folgenden Erklaerungen stellvertretend fuer alle moeglichen Definiere-Byte-Pseudooperationen verwendet.

DB wird verwendet, um einen Wert (Kette oder numerisch) in einem Speicherplatz abzulegen, beginnend ab laufenden Speicherplatzzuordnungszaehler.
Ausdruecke muessen sich auf ein Byte auswerten lassen. Wenn das hoehere Byte Ø oder 255 ergibt, ergibt sich kein Fehler, anderenfalls wird ein A-Fehler erzeugt.

Ketten mit 3 oder mehr Zeichen koennen in Ausdruecken nicht verwendet werden (sie muessen unmittelbar von einem Komma oder dem Ende der Zeile gefolgt werden).
Die Zeichen einer 8080- oder Z80-Kette werden in der Reihenfolge ihres Auftretens gespeichert, jedes als ein Ein-Byte-Wert, das hoechstwertige Bit auf Ø gesetzt.
Beispiel:

```
                    DB   'AB'
                    DB   'AB' AND ØFFH
                    DB   'ABC'
```

erzeugt:

```
ØØØØ' 41 42      DB  'AB'
ØØØ2' 42         DB  'AB' AND ØFFH
ØØØ3' 41 42 43   DB  'ABC'
```

Definiere Zeichen

```
DC <string>
```

DC speichert die Zeichen der Kette in <string> in aufeinanderfolgende Speicherplaetze, beginnend ab laufenden Speicherzuweisungszaehler.
Wie bei DB werden die Zeichen in der Reihenfolge ihres Auftretens gespeichert, jedes als ein Ein-Byte-Wert, das hoechstwertige Bit auf Ø gesetzt. Jedoch wird beim letzten Zeichen der Kette das hoechstwertige Bit auf 1 gesetzt.
Es wird ein Fehler erzeugt, wenn das Argument von DC eine Nullkette ist.

Beispiel:

```
                    MARKE:  DC  "ABC"
```

erzeugt:

```
ØØØØ  41 42 C3   MARKE:  DC  "ABC"
```

Definiere Speicher

```
DS   <exp>[,<val>]
DEFS <exp>[,<val>]
```

Die Pseudooperation "Definiere Speicher" reserviert einen Speicherbereich. Der Wert von <exp> gibt die Anzahl der zu reservierenden Bytes an.
Zum Initialisieren des reservierten Bereiches setzt <val> den gewuenschten Wert.

Wenn <val> weggelassen ist, wird der reservierte Speicher nicht initialisiert. Der reservierte Speicherbereich wird nicht automatisch auf ØØH initialisiert. Eine Moeglichkeit, den reservierten Speicherbereich auf ØØH zu initialisieren, bietet der /M-Schalter waehrend der Uebersetzung (siehe Pkt. 9.5.2.4.).

Alle Namen, die in Ausdruecken verwendet werden, muessen vorher definiert sein (alle Namen muessen im Pass 1 zu diesem Zeitpunktpunkt bekannt sein).
Ansonsten wird waehrend des Pass 1 ein V-Fehler erzeugt, ein U-Fehler kann im 2. Pass erzeugt werden; ein Phasenfehler ergibt sich wahrscheinlich, weil die Pseudooperation "Definiere Speicher" im Pass 1 keinen Code erzeugt hat.

Beispiel:

```
DS  1ØØH
```

reserviert 1ØØH Byte Speicherplatz nicht initialisiert, d.h., es bleiben dort die Werte stehen, die an dieser Stelle waren, bevor das Programm geladen wurde.

Soll der Bereich auf ØØH initialisiert werden, kann waehrend der Uebersetzung der /M-Schalter verwendet werden.
Oder es kann die folgende Anweisung geschrieben werden, um den reservierten Speicherbereich auf ØØH oder einen beliebigen anderen Wert zu setzen.

```
DS  1ØØH,2
```

reserviert 1ØØH Bytes, von denen jedes Byte mit dem Wert 2 (Ø2H) initialisiert wird.

Definiere Wort

```
DW    <exp>[,<exp>...]
DEFW  <exp>[,<exp>...]
```

Die Pseudooperation "Definiere Wort" speichert den Wert des Ausdruckes in aufeinanderfolgende Speicherplaetze, beginnend ab laufenden Speicherzuordnungszaehler. Ausdruecke sind 2-Byte Werte (Wort).
Die Werte werden, das niederwertige Byte zuerst und dann das hoeherwertige Byte, gespeichert.Das ist ein Unterschied zu DB!

Beispiel:

```
                MARKE:  DW      1234H
```

erzeugt:

```
ØØØØ 1234   MARKE:  DW      1234H
```

Bemerkung: Die Bytes werden in der Liste in Reihenfolge ihres Auftretens gezeigt, nicht in der Reihenfolge, wie sie abgespeichert sind.

Equate

```
<name> EQU <exp>
```

EQU weist <name> den Wert des Ausdruckes <exp> zu. <name> kann ein Label, ein Symbol oder eine Variable sein und spaeter in Ausdruecken verwendet werden.
Nach <name> darf kein Doppelpunkt(e) stehen.
Wenn <exp> ein External ist, wird ein Fehler erzeugt.
Wenn <name> schon einen anderen Wert als <exp> hat, erscheint ein M-Fehler.
Wenn <name> spaeter im Programm wieder (zurueck) definiert werden soll, so ist die Pseudooperation SET oder ASET anstelle von EQU zu verwenden.
Das ist ein Unterschied zu SET.

Beispiel:

```
BUF EQU ØF3H
```

Externes Symbol

```
EXT             <name>[,<name>...]
EXTRN           <name>[,<name>...]
EXTERNAL        <name>[,<name>...]
BYTE EXT        <symbol>
BYTE EXTRN      <symbol>
BYTE EXTERNAL   <symbol>
```

Diese Pseudooperation erklaert, dass der (die) Name(n) in der

Liste extern (ist) sind (d.h. definiert in einem anderen Modul).
Ist einer der in der Liste auftretenden Namen im laufenden Programm definiert, wird ein M-Fehler erzeugt.
Ein Verweis auf einen Namen, bei dem dem Namen unmittelbar 2 Doppelkreuze folgen (z.B. NAME##) wird ebenfalls als EXTERNAL erklaert.
EXTERNAL's koennen auf ein oder zwei Byte ausgewertet werden.
Bei allen externen Symbolnamen werden nur die ersten 6 Zeichen an den Programmverbinder uebergeben.
Zusaetzliche Zeichen werden intern abgeschnitten.

Beispiel:

```
    EXTRN       MARKEX
```

Der Assembler generiert fuer diese Anweisungszeile keinen Code beim Uebersetzten dieses Moduls. Wenn MARKEX als Argument in einer CALL-Anweisung verwendet wird, wird nur der Code fuer CALL erzeugt und fuer MARKEX wird ein Ø-Wert eingetragen.
Der Programmverbinder durchsucht alle geladenen Moduln nach einer PUBLIC MARKEX-Anweisung und verwendet die Definition, die er fuer MARKEX gefunden hat in der CALL MARKEX-Anweisung.

PUBLIC-Symbol

```
    ENTRY     <name>[,<name>...]
    GLOBAL    <name>[,<name>...]
    PUBLIC    <name>[,<name>...]
```

Die Pseudooperation PUBLIC erklaert jeden Namen der Liste als intern und deshalb verfuegbar fuer die Verwendung in diesem Programm und in anderen, die gemeinsam geladen und mit dem Programmverbinder gebunden sind.
Es wird ein M-Fehler erzeugt, wenn der Name ein EXTERNAL oder ein Common-Block-Name ist.
Nur die ersten 6 Zeichen des PUBLIC-Symbolnamens werden an den Programmverbinder uebergeben.
Zusaetzliche Zeichen werden intern abgeschnitten.

Beispiel:

```
            PUBLIC      MARKEX
MARKEX:     LD          HL,BER1
```

Der Assembler uebersetzt die LD-Anweisung wie ueblich, aber er generiert fuer die PUBLIC MARKEX-Anweisung keinen Code.
Wenn der Programmverbinder im anderen Modul die EXTERN MARKEX-Anweisung findet, weiss er, dass er suchen muss, bis er diese PUBLIC MARKEX-Anweisung gefunden hat. Dann verbindet der Programmverbinder den Adresswert der MARKEX:LD HL,BER1-Anweisung mit der (den) CALL MARKEX-Anweisung(en) im (in den) anderen Modul(n).

SET

```
<name>  SET   <exp>     (nicht fuer den .Z80-Modus)
<name>  DEFL  <exp>
<name>  ASET  <exp>
```

Die Pseudooperation SET weist <name> den Wert von <exp> zu. <name> kann ein Label, ein Symbol oder eine Variable sein und kann spaeter in Ausdruecken verwendet werden.

Nach <name> duerfen keine Doppelpunkte stehen. Wenn <exp> ein EXTERNAL ist, wird ein Fehler erzeugt.

Die Pseudooperation SET kann im .Z80-Modus nicht verwendet werden, weil SET ein Z80-Operationscode ist.

ASET und DEFL koennen in beiden Anweisungslisten verwendet werden.
Wenn <name> spaeter neu definiert werden soll, muss eine der SET-Pseudooperationen anstelle von EQU verwendet werden.
Zum neu Definieren von <name> kann jede beliebige SET-Pseudooperation benutzt werden, unabhaengig, mit welcher Pseudooperation die urspruengliche Definition erfolgte (das Verbot von SET im .Z80-Modus wird davon nicht beruehrt).
Das ist ein Gegensatz zu EQU.

Beispiel:

```
MARKE   ASET  BER+1ØØØH
```

Wann immer MARKE als Ausdruck (Operand) verwendet wird, wertet der Assembler den Ausdruck BER+1ØØØH aus und setzt diesen Wert fuer MARKE. Spaeter, wenn MARKE einen anderen Wert darstellen soll, wird einfach eine MARKE ASET-Anweisung mit einem anderen Ausdruck eingetragen.

```
MARKE   ASET   BER+1ØØØH
   .
   .
   .
MARKE   ASET   3ØØØH
   .
   .
   .
MARKE   DEFL   6CDEH
```

9.4.1.3. Pseudooperation zum Zuweisen des Speicherplatzzuordnungszaehler-Modus

Viele Pseudooperationen beziehen sich auf den aktuellen Speicherplatzzuordnungszaehler.
Der laufende Speicherplatzzuordnungszaehler ist die Adresse des naechsten Bytes, das zu erzeugen ist.
Im Assembler erhalten Symbole und Ausdruecke ihren

zugewiesenen Modus (siehe Pkt. 9.3.2.)

Jeder Modus fuer ein Speichersegment wird durch den Programmverbinder entsprechend dem Typ der uebersetzten Anweisung zugeordnet.

Es gibt 4 Modi:

- absoluter Modus
- datenrelativer Modus
- coderelativer Modus
- commonrelativer Modus.

Wenn der Speicherplatzzuordnungszaehler-Modus absolut ist, dann ist er eine absolute Adresse.

Wenn der Speicherplatzzuordnungszaehler-Modus relativ ist, dann ist er eine relative Adresse und kann einen Offset einbeziehen von der absoluten Startadresse eines relativen Segments, das vom Programmverbinder geladen werden kann.

Die Pseudooperation zur Zuweisung des Speicherplatzzuordnungszaehler-Modus wird fuer die Spezifizierung genutzt, in welchem Typ ein Programmteil zu uebersetzen ist.

Absolutes Segment

ASEG

ASEG hat niemals Operanden. ASEG erzeugt nichtverschieblichen Code.
ASEG setzt den Speicherplatzzuordnungszaehler auf ein absolutes Speichersegment (aktuelle Adresse). Der Standardwert ist Ø, das kann die Ursache sein, dass Teile des Betriebssystems ueberschrieben werden.
Nach ASEG sollte eine ORG-Anweisung mit 1Ø3H oder hoeher geschrieben werden.

Code-Segment

CSEG

CSEG hat niemals Operanden.
Der Code wird im coderelativen Modus uebersetzt und kann in den ROM oder PROM geladen werden.
CSEG setzt den Speicherzuordnungszaehler auf das coderelative Segment des Speichers zurueck.
Die Zuordnung ist diejenige des letzten CSEG (Standard Ø), wenn nicht danach eine ORG-Anweisung folgt, die die Zuordnung aendert.

Man muss jedoch beachten, dass die ORG-Anweisung im CSEG-Modus keine absolute Adresse setzt.
Eine ORG-Anweisung im CSEG-Modus veranlasst den Assembler, zu

der letzten fuer CSEG geladenen Adresse die im Argument der ORG-Anweisung festgelegte Anzahl Bytes zu addieren.

Wenn beispielsweise ORG 5Ø angegeben ist, addiert der Assembler 5Ø Bytes zu der aktuellen CSEG-Zuordnung, und dies ist dann die neue Zuordnung von CSEG.
Die Wirkung der auf ein CSEG (oder DSEG) folgenden ORG-Anweisung ist die, dass einem Modul ein Offset gegeben werden kann. ORG setzt keine absolute Adresse fuer CSEG, sondern CSEG behaelt seine Verschieblichkeit.

Will man fuer CSEG eine absolute Adresse setzen, verwendet man den /P-Schalter im Programmverbinder.

CSEG ist der Standard-Modus des Assemblers. Der Assembler beginnt automatisch mit einem CSEG und der Speicherplatzzuordnungszaehler im coderelativen Modus weist auf den Platz Ø des coderelativen Speichersegments. Alle folgenden Anweisungen werden in das coderelative Speichersegment uebersetzt, bis eine ASEG-, DSEG- oder COMMON- Pseudooperation abgearbeitet wird.

CSEG wird folglich eingetragen, damit der Assembler zum coderelativen Modus zurueckkehrt, der Speicherplatzzuordnungszaehler wird auf den Punkt des naechsten freien Platzes im coderelativen Segment zurueckgesetzt.

Daten-Segment

DSEG

Die Pseudooperation DSEG hat keine Operanden.
DSEG spezifiziert Segmente des uebersetzten verschieblichen Codes, die spaeter nur in den RAM geladen werden.

DSEG setzt den Speicherplatzzuordnungszaehler auf das datenrelative Speichersegment.
Die Zuordnung des datenrelativen Zaehlers ist die des letzten DSEG (Standard Ø), wenn nicht eine folgende ORG-Anweisung die Zuordnung aendert.
Man muss jedoch beachten, dass die ORG-Anweisung im DSEG-Modus keine absolute Adresse setzt.

Eine ORG-Anweisung im DSEG-Modus veranlasst den Assembler, zu der letzten fuer DSEG geladenen Adresse die im Argument der ORG-Anweisung festgelegte Anzahl Bytes zu addieren.
Wenn beispielsweise ORG 5Ø angegeben ist, addiert der Assembler 5Ø Bytes zu der aktuellen DSEG-Zuordnung, und dies ist dann die neue Zuordnung von DSEG.

Die Wirkung der auf ein DSEG (oder CSEG) folgenden ORG-Anweisung ist die, dass einem Modul ein Offset gegeben werden kann. ORG setzt keine absolute Adresse fuer DSEG, sondern DSEG behaelt seine Verschieblichkeit.
Will man eine absolute Adresse fuer DSEG setzten, verwendet

man den /D-Schalter im Programmverbinder.

Common-Block (gemeinsamer Bereich)

COMMON /[<blockname>]/

das Argument von COMMON ist der Name des gemeinsamen Bereiches.
COMMON legt einen gemeinsamen Datenbereich fuer alle COMMON-Bloecke an, die in dem Programm benannt sind.
Wenn <blockname> weggelassen wurde oder aus Leerzeichen besteht, wird der Block als leerer Bereich betrachtet.

COMMON-Anweisungen sind nicht ausfuehrbare Speicherzuordnungsanweisungen.
COMMON weist einem Speicherbereich, genannt gemeinsamer Bereich, Variable, Felder und Daten zu.
Das ermoeglicht, dass sich verschiedene Programmoduln in denselben Speicherbereich teilen.

Die der COMMON-Anweisung folgenden Anweisungen werden in den COMMON-Bereich unter <blockname> uebersetzt.
Die Laenge eines COMMON-Bereiches ergibt sich aus der Anzahl Bytes, die erforderlich sind, um die in diesem COMMON-Block definierten Variablen, Felder und Daten unterzubringen. Der COMMON-Block wird beendet, wenn eine andere Pseudooperation zur Zuweisung des Speicherplatzzuordnungszaehler-Modus auftritt.
Gemeinsame Bloecke desselben Namens koennen unterschiedliche Laenge haben.
Ist die Laenge verschieden, dann muss der Programmodul mit dem laengsten gemeinsamen Block zuerst geladen werden (, d.h., er muss der erste Modulname in der Kommandozeile des Programmverbinders sein).
Siehe dazu auch die Beschreibung des Programmverbinders.

COMMON setzt den Speicherplatzzuordnungszaehler auf den ausgewaehlten gemeinsamen Speicherblock.
Die Zuweisung ist immer der Beginn des Bereiches, so dass Kompatibilitaet mit der FORTRAN COMMON-Anweisung besteht.

Beispiel:

```
        COMMON      /DATBIN/
BER2    EQU         1ØØH
        DB          ØFFH
        DW          1234H
        DC          'WERK'
        CSEG
```

Set Origin

ORG <exp>

Der Wert des Speicherplatzzuordnungszaehlers kann zu jeder Zeit durch die Verwendung von ORG geaendert werden. Im ASEG-Modus wird der Speicherplatzzuordnungszaehler auf den Wert von <exp> gesetzt, und der Assembler weist den erzeugten Code beginnend mit diesem Wert zu.

Im CSEG-, DSEG- und COMMON- Modus wird der Speicherplatzzuordnungszaehler in diesem Segment um den Wert von <exp> erhoeht, und der Assembler weist den erzeugten Code beginnend mit der letzten geladenen Segmentadresse plus dem Wert von <exp> zu.

Alle in <exp> verwendeten Namen muessen im Pass 1 bekannt sein und der Wert muss entweder absolut sein oder im selben Bereich, wie der Speicherplatzzuordnungszaehler.

Beispiel:

```
.DSEG
ORG   5Ø
```

Setzt den datenrelativen Speicherplatzzuordnungszaehler auf 5Ø, relativ zum Start des datenrelativen Speichersegments. Diese Methode liefert Verschieblichkeit. Die ORG <exp> - Anweisung spezifiziert im CSEG- oder DSEG- Modus keine feste Adresse, vielmehr laedt der Programmverbinder das Segment auf eine flexible Adresse, die fuer die gemeinsam zu ladenden Moduln verwendet wird.
Andererseits wird ein Programm, das mit den Anweisungen

```
ASEG
ORG 8ØØH
```

beginnt und vollstaendig im absoluten Modus uebersetzt ist, immer beginnend ab 8ØØH geladen, es sei denn, die ORG-Anweisung wird in der Quelldatei geaendert.
Das heisst, die dem ASEG folgende ORG <exp> - Anweisung setzt das Segment auf eine feste (im Beispiel absolute) Adresse, die durch <exp> bestimmt wird.

Das gleiche Programm - im coderelativen Modus, ohne ORG - Anweisung uebersetzt - kann auf jede beliebige Adresse geladen werden, wenn an die Kommandokette des Programmverbinders der Schalter /P:<address> angehaengt wird.

Verschieben

```
.PHASE <exp>
.DEPHASE
```

.PHASE ermoeglicht es, Code in einen Bereich zu laden, aber nur in einem anderen Bereich mit der durch <exp> spezifi-

zierten Adresse abzuarbeiten.
<exp> muss ein absoluter Wert sein. .DEPHASE wird verwendet, um anzuzeigen, dass der verschobene Codeblock zu Ende ist.
Der Modus innerhalb eines .Phase-Blockes ist absolut. Der Code wird in den Bereich geladen, wenn die .PHASE-Anweisung erreicht wird.
Der Code innerhalb eines Blockes wird spaeter auf die durch <exp> fuer die Ausfuehrung spezifizierte Adresse transportiert.

Beispiel:

```
                        .PHASE      1ØØH
              MARKE:    CALL        UP1
                        JMP         MARKE1
                UP1:    RET
                        .DEPHASE
              MARKE1:   JMP         5
```

erzeugt folgende Uebersetzung:

```
                                    .PHASE      1ØØH
Ø1ØØ  CD Ø1Ø6 MARKE:    CALL        UP1
Ø1Ø3  C3 ØØØ7           JMP         MARKE1
Ø1Ø6  C9          UP1:  RET
                        .DEPHASE
ØØØ7' C3 ØØØ5 MARKE1:   JMP         5
                        END
```

.PHASE-.... .DEPHASE-Block sind eine Moeglichkeit, den Codeblock ab einer spezifischen absoluten Adresse auszufuehren.

9.4.1.4. Dateibezogene Pseudooperationen

Die dateibezogenen Pseudooperationen:

- fuegen lange Kommentare ein
- geben dem Modul einen Namen
- beenden den Modul oder
- transportieren andere Dateien in das laufende Programm

Comment-Anweisung

```
.COMMENT <delim><text><delim>
```

Das erste nichtleere Zeichen, das nach .COMMENT gefunden wird, wird als Begrenzer genommen. Der dem Begrenzer folgende <text> wird ein Kommentarblock, der fortgesetzt wird bis zum naechsten Auftreten des Begrenzers <delim>.
.COMMENT wird fuer lange Kommentare verwendet. Es ist nicht notwendig, das Semikolon zum Anzeigen des Kommentars zu schreiben.

Waehrend der Uebersetzung wird der .COMMENT-Block ignoriert und nicht uebersetzt.

Beispiel:

```
.COMMENT * hier kann jeder beliebige Text
           eingetragen sein
           .
           .
           .
         * ; Rueckkehr zur normalen Uebersetzung
```

Programmende

END [<exp>]

Die End-Anweisung bezeichnet das Programmende. Wenn diese Anweisung fehlt, wird folgende Warnung erzeugt:

"%NO END statement".

<exp> kann ein Label, ein Symbol, eine Zahl oder jedes andere gueltige Argument sein, das der Programmverbinder als Startpunkt fuer das Pogramm laden kann.

Wenn <exp> angegeben wurde, schreibt der Programmverbinder auf die Adresse 1ØØH eine Sprunganweisung im 8080-Code zu der in <exp> angegebenen Adresse.
Wenn <exp> nicht angegeben wurde, wird an den Programmverbinder fuer dieses Programm keine Startadresse uebergeben und die Ausfuehrung beginnt mit dem ersten geladenen Modul.
(Das heisst, wenn kein <exp> spezifiziert wurde, ist der /G-Schalter im Programmverbinder unwirksam.)
<exp> teilt dem Programmverbinder mit, dass das Programm ein Hauptprogramm ist, sonst wird es als Unterprogramm angesehen.
Wenn nur Assemblerprogramme gebunden werden, von denen keines eine END-Anweisung mit <exp> enthaelt, wird der Programmverbinder nach dem Hauptprogramm fragen.
Haben zwei oder mehr Programme eine END-Anweisung mit <exp>, so kann der Programmverbinder nicht unterscheiden, welches das Hauptprogramm ist.

Wenn zwei oder mehr Programme gebunden werden sollen, muss /G:Name- oder /E:Name-Schalter im Programmverbinder benutzt werden.
Name ist der <exp> der END-Anweisung des Programms, welches als Hauptprogramm angesehen werden soll.
Wenn ein Programm in einer hoeheren Programmiersprache gemeinsam mit einem Assemblerprogramm geladen werden soll, wird vom Programmverbinder automatisch das Programm in der hoeheren Sprache als Hauptprogramm genommen.
Das kann man aendern mit dem /G:Name- oder /E:Name-Schalter, dabei muss Name der Name des Assemblerprogramms sein.

Eine andere Moeglichkeit ist das Setzen eines CALL oder INCLUDE an den Anfang des Programms in der hoeheren Programmiersprache, um das Assemblerprogramm zuerst abzuarbeiten.

INCLUDE-Anweisung

```
INCLUDE   <dateiname>
□INCLUDE  <dateiname>
MACLIB    <dateiname>
```

Alle drei Pseudooperationen sind sinnverwandt.
Waehrend der Assemblierung fuegen die INCLUDE-Pseudooperationen Quelloode aus einer anderen Quelldatei in die laufende Quelldatei ein.

Durch die Verwendung der INCLUDE-Pseudooperation ist es nicht erforderlich, haeufig gebrauchte Anweisungsfolgen in der laufenden Quelldatei wiederholt zu schreiben.

<dateiname> ist jede fuer das Betriebssystem gueltige Dateispezifikation und muss in grossen Buchstaben geschrieben werden.
Wenn der Dateityp und/oder die Geraetezuweisung andere als die zugewiesenen sind, muss die Quelldatei-Spezifikation diese enthalten.
Der zugewiesene Dateityp fuer Quelldateien ist .MAC. Die Standard-Geraetezuweisung ist das aktuelle (current) Laufwerk bzw. Geraet.

Die INCLUDE-Datei wird eroeffnet und in die aktuelle Quelldatei uebersetzt, unmittelbar anschliessend an die INCLUDE-Anweisung. Wenn das Dateiende erreicht worden ist, setzt der Assembler mit der auf INCLUDE folgenden Anweisung fort.
Geschachtelte INCLUDE's sind nicht erlaubt. Fuer den Fall, dass dies auftritt, wird ein Objektcode-Syntax-Fehler, "O", erzeugt.
Die im Operandenfeld spezifizierte Datei muss existieren, ansonsten wird ein "V"-Fehler erzeugt und das INCLUDE wird ignoriert. Es wird auch ein "V"-Fehler erzeugt, wenn der Dateityp der INCLUDE-Datei nicht .MAC ist.

Im Assemblerprotokoll wird der Buchstabe "C" zwischen den uebersetzten Code und die Quellzeile gedruckt (Siehe Pkt. 9.5.3.).

Modulname

```
NAME ("modulname")
```

Name definiert den Namen fuer den Modul. Die runden Klammern und Anfuehrungszeichen um den Modulnamen sind erforderlich. Nur die ersten 6 Zeichen des Modulnamens sind signifikant.
Ein Modulname kann auch mit der Pseudooperation TITLE definiert werden. Wenn keine der beiden Pseudooperationen NAME

oder TITLE vorhanden sind, wird der Modulname aus dem Namen der Quellcodedatei erzeugt.

RADIX (Zahlenbasis)

.RADIX <exp>

<exp> in einer .RADIX-Anweisung ist immer eine dezimale numerische Konstante, ohne Ruecksicht auf die aktuelle Zahlenbasis.
Die .RADIX-Anweisung erlaubt die Aenderung der Standardzahlenbasis in irgendeine beliebige zwischen 2 und 16.

.RADIX aendert nicht die Zahlenbasis auf der Liste, sondern ermoeglicht die Eingabe numerischer Werte in der gewuenschten Basis ohne spezielle Schreibweise. (Werte in anderen Zahlenbasen erfordern spezielle Schreibweisen, erlaeutert im Abschnitt "Operanden")

Die Werte im erzeugten Code erscheinen immer in der hexadezimalen Zahlenbasis.

Beispiel:

```
DEC:    DB      2Ø
        .RADIX  2
BIN:    DB      ØØØ1111Ø
        .RADIX  16
HEX:    DB      ØCF
       .RADIX   8
OCT:    DB      73
       .RADIX   1Ø
DECI:   DB      16
HEXA:   DB      ØCH
```

erzeugt folgende Uebersetzung:

```
ØØØØ' 14  DEC:    DB      2Ø
ØØØ2              .RADIX  2
ØØØ1' 1E  BIN:    DB      ØØØ1111Ø
ØØ1Ø              .RADIX  16
ØØØ2' CF  HEX:    DB      ØCF
ØØØ8              .RADIX  8
ØØØ3' 3B  OCT:    DB      73
ØØØA              .RADIX  1Ø
ØØØ4' 1Ø  DECI:   DB      16
ØØØ5' ØC  HEXA:   DB      ØCH
```

Request (Anforderung)

.REQUEST <dateiname>[,<dateiname>...]

.REQUEST stellt an den Programmverbinder die Forderung, die in der Liste enthaltenen Dateinamen nach undefinierten

externen Symbolen zu durchsuchen (externe Symbole, fuer die zum gleichen Zeitpunkt kein korrespondierendes PUBLIC-Symbol geladen ist).
Die vom Programmverbinder gefundenen undefinierten Symbole werden benoetigt, um einen oder mehrere zusaetzliche Moduln fuer einen kompletten Programmverbinderlauf zu laden.

Die Dateinamen in der Liste sollten gueltige Symbole sein und weder Dateityp noch Geraetezuweisung beinhalten.
Der Programmverbinder setzt den Standarddateityp .REL und das aktuelle Laufwerk voraus.

Beispiel:

```
      .
      .
      .
   .REQUEST SUBR1
      .
      .
      .
```

Der Programmverbinder durchsucht SUBR1 nach externen Symbolen, fuer die kein korrespondierendes PUBLIC-Symbol in den bereits geladenen Moduln definiert ist.

9.4.1.5 Pseudooperationen zur Listensteuerung

Die Pseudooperationen zur Listensteuerung fuehren 2 allgemeine Funktionen aus:

- Formatsteuerung
- Listensteuerung.

Die Pseudooperationen zur Formatsteuerung ermoeglichen dem Programmierer Seitenwechsel und Kopfzeilen einzufuegen.
Das Protokollieren gesamter Assemblerdateien oder von Teilen davon wird mit den Pseudooperationen zur Listensteuerung ein- und ausgeschaltet.

Pseudooperationen zur Formatsteuerung

Diese Pseudooperationen erlauben das einfuegen von Seitenwechseln und das Festlegen von Titel- und Untertitelzeilen im Assemblerprotokoll.

Form Feed

```
*EJECT [<exp>]
PAGE   [<exp>]
¤EJECT
```

Die Pseudooperation Seitenwechsel veranlasst den Assembler, die Ausgabe mit einer neuen Seite zu beginnen.
Er setzt ein Seitenwechsel-Steuerzeichen an das Seitenende in die Protokolldatei.
Der Wert von <exp>, falls er angegeben wurde, bestimmt den Umfang der neuen Seite (gemessen in Zeilen pro Seite) und muss im Bereich zwischen 1Ø und 255 liegen. Die Standardzeilenzahl pro Seite ist 5Ø.
*EJECT muss in der Spalte 1 beginnen.

Beispiel:

```
*EJECT 72
```

Der Assembler veranlasst den Drucker, eine neue Seite zu beginnen und jedesmal 72 Programmzeilen zu drucken (Diese Angabe ist zu waehlen bei einer Papierlaenge von 12" und 6 Zeilen pro Zoll).

Titel

```
TITLE <text>
```

TITLE spezifiziert eine Kopfzeile, die auf jede Seite als erste Zeile gesetzt wird.
Wenn mehr als eine TITLE-Anweisung angegeben wird, wird ein "Q"-Fehler erzeugt.
Wenn keine NAME-Anweisung geschrieben wurde, werden die ersten 6 Zeichen des Titels als Modulname verwendet.
(Ist weder NAME noch TITLE vorhanden, wird der Modulname aus der Quelldatei genommen.)

Beispiel:

```
TITLE PROG1
```

Der Modul heisst jetzt PROG1. Er kann mit diesem Namen aufgerufen werden und PROG1 wird als Kopf auf jede Protokollseite gedruckt.

Untertitel

```
SUBTTL  <text>
¤TITLE (<text>)
```

SUBTTL spezifiziert einen Untertitel, der in jeden Seitenkopf in die Zeile nach dem Titel gedruckt werden soll.

<text> wird abgebrochen, wenn 60 Zeichen erreicht wurden.
In einem Programm kann eine beliebige Anzahl von Untertiteln angegeben werden.
Wenn der Assembler eine SUBTTL-Anweisung findet, wird der alte Text durch den neuen Text ersetzt.
Um den SUBTTL fuer einen Teil der Ausgabe auszuschalten, muss ein SUBTTL mit einer leeren Kette fuer <text> geschrieben werden.

Beispiel:

```
SUBTTL SPEZIELLE E/A-ROUTINE
  .
  .
  .
SUBTTL
  .
  .
  .
  .
```

Das erste SUBTTL veranlasst den Druck von SPEZIELLE E/A-ROUTINE im Kopf jeder Seite.
Das zweite SUBTTL schaltet den Untertitel aus (die Untertitelzeile im Protokoll besteht aus Leerzeichen).

Allgemeine Pseudooperationen zur Listensteuerung

```
.LIST   Protokollieren aller Zeilen mit ihrem Code
.XLIST  Unterdruecken des Protokollierens
```

.LIST ist die Standardbedingung.
Wenn eine Protokolldatei in der Kommandozeile spezifiziert wurde, dann wird die Datei aufgelistet.
Wenn .XLIST in der Quelldatei vorgefunden wird, werden Quell- und Objektcode nicht aufgelistet.
.XLIST wirkt bis zum Auftreten eines .LIST.
.XLIST setzt alle anderen Pseudooperationen zur Listensteuerung (ausser .LIST) ausser Kraft.

Beispiel:

```
  .
  .
  .
.XLIST      ; Protokollieren wird hier abgebrochen
  .
  .
  .
  .
.LIST       ; Protokollieren wird hier fortgestzt
  .
  .
  .
```

Ausgeben einer Meldung auf den Bildschirm

```
.PRINTX <delim><text><delim>
```

Das erste nichtleere Zeichen nach .PRINTX ist der Begrenzer. Der folgende Text wird waehrend der Uebersetzung auf den Bildschirm ausgegeben bis der naechste Begrenzer auftritt.
.PRINTX ist nuetzlich, um den Prozess waehrend einer langen Uebersetzung darzustellen oder den Wert von Schaltern der bedingten Assemblierung anzuzeigen.
.PRINTX wird in beiden Paessen ausgegeben.
Wird die Ausgabe nur in einem Pass gewuenscht, verwendet man die Pseudooperation IF1 oder IF2, um zu bestimmen, in welchem Pass die Ausgabe gewuenscht wird (siehe auch Pkt. 9.4.3.).

Beispiel:

```
PRINTX *Uebersetzung zur Haelfte fertig*
```

Der Assembler gibt diese Meldung bei ihrem Auftreten auf den Bildschirm aus.

```
IF1
.PRINTX *PASS 1 BEENDET*; Meldung nur im Pass 1
ENDIF
IF2
.PRINTX *PASS 2 BEENDET*; Meldung nur im Pass 2
ENDIF
```

Pseudooperationen zur bedingten Listensteuerung

Die 3 Pseudooperationen zur bedingten Listensteuerung werden verwendet, um anzuzeigen, ob in einem falschen Bedingungsblock enthaltene Anweisungen auf der Liste erscheinen oder nicht.
Siehe dazu auch die Beschreibung zum /X-Schalter im Abschnitt "Schalter".

Unterdruecken falscher Bedingungen

```
.SFCOND
```

.SFCOND unterdrueckt den Teil der Liste, welcher bedingte Ausdruecke beinhaltet, die als falsch gewertet werden.

Protokollieren

```
.LFCOND
```

.LFCOND sichert das Protokollieren der bedingten Ausdruecke, die als falsch gewertet wurden.

Umkippen der Listensteuerung falscher Bedingungen

.TFCOND

.TFCOND kippt die aktuelle Einstellung um, die durch das Vorhandensein oder Nichtvorhandensein vom /X-Schalter in der Kommandozeile gesetzt wurde.
.TFCOND ist unabhaengig von .LFCOND und .SFCOND.
Wenn /X vorhanden war, veranlasst .TFCOND, die falschen Bedingungen zu drucken. Wenn /X nicht angegeben wurde, unterdrueckt .TFCOND die falschen Bedingungen.

Pseudooperationen zur Listensteuerung der Makro-Erweiterung

Pseudooperationen zur Listensteuerung der Makro-Erweiterung steuern die Protokollierung der Zeilen innerhalb der Erweiterung eines Makros oder einer Wiederholungs-Pseudooperation (REPT, IRP, IRPC).

Ausschliessen von Makrozeilen, die keinen Code erzeugen

.XALL

.XALL ist Standard. .XALL protokolliert Quellcode und Objektcode, der durch einen Makro erzeugt wird.
Quellzeilen, die keinen Code erzeugen, werden nicht protokolliert.

Protokollieren Makrotext

.LALL

.LALL protokolliert den kompletten Text fuer die ganze Macroerweiterung einschliesslich der Zeilen, die keinen Code erzeugen.

Unterdruecken der Makro-Protokollierung

.SALL

.SALL unterdrueckt das Protokollieren des gesamten Textes der Makroerweiterung und des durch den Makro erzeugten Objektcodes.

Pseudooperationen zur Steuerung der Cross-Referenz

Es ist moeglich, die Cross-Referenz nur fuer einen Teil, aber nicht fuer das ganze Programm zu generieren. Dazu verwendet man die Pseudooperation .CREF und .XCREF in der Quelldatei fuer den Assembler.

Diese beiden Pseudooperationen koennen an jede beliebige Stelle im Programm in das Operationsfeld geschrieben werden.
Wie alle Pseudooperationen zur Listensteuerung haben sie keine Argumente.

Unterdruecken der Cross-Referenz

.XCREF

.XCREF schaltet .CREF (Standard) aus. .XCREF bleibt wirksam bis der Assembler ein .CREF trifft.
.XCREF wird verwendet, um in einem ausgewaehlten Teil der Dateien das Erzeugen einer Cross-Referenz zu unterdruecken.
Weil weder .CREF noch .XCREF Wirkung haben, wenn nicht der /C-Schalter in der Assembler-Kommandozeile gesetzt ist, muss .XCREF nicht verwendet werden, wenn eine normale Liste (ohne Cross-Referenz) gewuenscht wird. Das wird auch erreicht, wenn /C in der Kommandozeile weggelassen wird.

Erzeugen einer Cross-Referenz

.CREF

.CREF ist die Standard-Bedingung.
.CREF wird verwendet, um mit der Erstellung der Cross-Referenz fortzufahren, nachdem diese durch .XCREF gestoppt worden war.
.CREF bleibt wirksam bis ein .XCREF auftritt.
.CREF hat nur dann Wirkung, wenn der /C-Schalter in der Kommandozeile des Assemblers gesetzt ist.

9.4.2. Makrofaehigkeit

Die Makrofaehigkeit erlaubt, Bloecke von Anweisungen zu schreiben, die wiederholt verwendet werden koennen, ohne dass sie wieder aufgeschrieben werden muessen.
Diese Anweisungsbloecke beginnen entweder mit der Pseudooperation Makrodefinition oder mit einer der Wiederholungspseudooperationen. Sie enden mit ENDM. Alle Makro-Pseudooperationen koennen innerhalb eines Makroblockes verwendet werden.
Die Schachtelungstiefe von Makros ist nur durch den Speicherplatz begrenzt.
Die Makrofaehigkeit des Assemblers beinhaltet folgende Pseudooperationen, die nachfolgend erlaeutert werden:

- Makrodefinition

  MACRO

- Wiederholungen

```
REPT    (Wiederholung)
IRP     (undefinierte Wiederholung)
IRPC    (undefinierte Wiederholung mit Zeichen)
```

- Abschluss

```
ENDM
EXITM
```

- nur einmal vorkommende Symbole innerhalb des Makroblockes

```
LOCAL
```

Die Makrofaehigkeit unterstuetzt ausserdem einige spezielle Makrooperatoren:

```
&
;;
!
%
```

9.4.2.1. Pseudooperation zur Makrodefinition

```
<name> MACRO [<dummy>[,<dummy>...]]
        .
        .
        .
        ENDM
```

Der Anweisungsblock von MACRO bis ENDM bildet den Kern des Makros oder die Makrodefinition.
<name> ist ein LABEL und unterliegt den Regeln fuer die Bildung von Symbolen. <name> kann jede beliebige Laenge haben, aber nur die ersten 16 Zeichen werden an den Programmverbinder uebergeben.
Nachdem der Makro definiert worden ist, kann <name> zum Aufrufen des Makros verwendet werden.
<dummy> ist ein symbolischer Parameter, der durch den echten Parameter durch Eins-zu-Eins-Textsubstitution bei der Verwendung des Makros ersetzt wird.
Jedes <dummy> kann 32 Zeichen lang sein. Die Anzahl der symbolischen Parameter ist nur durch die Laenge der Zeile begrenzt.
Die symbolischen Parameter werden durch Kommas voneinander getrennt.
Der Assembler interpretiert alle Zeichen zwischen zwei Kommas als einen einzigen symbolischen Parameter.

Bemerkung:

Ein symbolischer Parameter wird ausschliesslich als solcher erkannt.

Wenn zum Beispiel ein Registername (A oder B) als symbolischer Parameter verwendet wurde, wird er waehrend der Erweiterung durch einen Parameter ersetzt.

Ein Makroblock wird nicht bei seinem Auftreten uebersetzt, sondern dann, wenn er aufgerufen wird, erweitert der Assembler die Makro-Aufruf-Anweisung durch den passenden Makroblock.
Wenn die Pseudooperation TITLE, SUBTTL oder NAME fuer den Teil des Programms, wo der Makroblock erscheint, verwendet werden soll, ist Vorsicht beim Schreiben dieser Anweisungszeile geboten.

Beispiel:

```
SUBTTL MACRO DEFINITION
```

Der Assembler uebersetzt die Anweisung als Makrodefinition mit dem Namen SUBTTL und DEFINITION als symbolischen Parameter. Um das zu vermeiden, koennte das Wort MACRO abgeaendert werden.

Makroaufruf

Soll ein Makro verwendet werden, muss eine Makroaufruf-Anweisung geschrieben werden.

```
<name> [<parameter>[,<parameter>...]]
```

<name> ist gleich <name> des Makroblockes.
<parameter> ersetzt Eins-zu-Eins <dummy>. Die Anzahl der Parameter wird nur durch die Laenge der Zeile begrenzt. Die Parameter muessen durch Kommas voneinander getrennt werden.
Wenn um Parameter, die durch Kommas getrennt sind, spitze Klammern geschrieben sind, uebergibt der Assembler alles, was in den spitzen Klammern steht, als einen einzigen Parameter.

Beispiel:

```
MAC 1,2,3,4,5
```

uebergibt 5 Parameter an den Makro, aber

```
MAC <1,2,3,4,5>
```

uebergibt nur einen Parameter.

Die Anzahl der Parameter in der Makroaufruf-Anweisung muss nicht mit der Anzahl der symbolischen Parameter (<dummy>) in der Makrodefinition uebereinstimmen.
Wenn der Aufruf mehr Parameter enthaelt als die Definition, werden die uebrigen ignoriert. Wenn es weniger sind, werden die uebriggebliebenen symbolischen Parameter Null gesetzt.
Nach jeder Makroaufruf-Anweisung wird der Makroblock in den uebersetzten Code eingefuegt.

Beispiel:

```
EXCHNG  MACRO     X,Y
        PUSH      X
        PUSH      Y
        POP       X
        POP       Y
        ENDM
```

Folgendes Programmstueck enthaelt den Makroaufruf:

```
        LDA       2FH
        MOV       H,A
        LDA       3FH
        MOV       D,A
        EXCHNG    H,D
```

Es wird folgende Uebersetzung erzeugt:

```
ØØØØ' 3A  ØØ2F      LDA       2FH
ØØØ3' 67            MOV       H,A
ØØØ4' 3A  ØØ3F      LDA       3FH
ØØØ7' 57            MOV       D,A
                    EXCHNG    H,D
ØØØ8' E5      +     PUSH      H
ØØØ9' D5      +     PUSH      D
ØØØA' E1      +     POP       H
ØØØB' D1      +     POP       D
```

9.4.2.2. Wiederholungs-Pseudooperationen

Die Pseudooperationen ermoeglichen es, Operationen in einem Codeblock so oft zu wiederholen, wie spezifiziert wurde.
Die Hauptunterschiede zwischen den Wiederholungs- und den Makropseudooperationen sind:

1. MACRO gibt dem Block einen Namen, mit welchem er in den Code an die Stelle, wo er gebraucht wird, aufgerufen werden kann.
   Der Makro kann in mehreren verschiedenen Programmen durch einfaches Schreiben einer Makroaufruf-Anweisung verwendet werden.

2. MACRO ermoeglicht das Uebergeben von Parametern an den Makroblock, wenn er aufgerufen wird; diese Parameter koennen sich aendern.

Die Wiederholungs-Pseudooperationen muessen als ein Teil des Codeblockes zugewiesen sein.
Das Verwenden der Wiederholungs-Pseudooperationen ist bequem, wenn die Parameter im Voraus bekannt sind und nicht geaendert werden und wenn die Wiederholung bei jeder Programmausfuehrung realisiert werden soll.
Mit der Makro-Pseudooperation muss der Makro jedesmal, wenn er

gebraucht wird, aufgerufen werden!
Beachte, dass jeder Wiederholungsblock mit der ENDM-Pseudooperation abgeschlossen werden muss.

Wiederholung

```
REPT <exp>
  .
  .
  .
ENDM
```

Wiederholen des Anweisungsblockes zwischen REPT und ENDM <exp>-mal.
<exp> ergibt eine vorzeichenlose 16-bit-Zahl.

Wenn <exp> ein externes Symbol oder undefinierte Operanden enthaelt, wird ein Fehler generiert.

Beispiel:

```
                X  SET  Ø
                   REPT 1Ø
                X  SET  X+1
                   DB   X
                   ENDM
```

erzeugt folgendes Uebersetzungsprotokoll:

```
ØØØØ            X  SET  Ø
                    REPT 1Ø
                 X  SET  X+1
                    REPT 1Ø
                    ENDM
ØØØØ 'Ø1    +      DB   X
ØØØ1 'Ø2    +      DB   X
ØØØ2 'Ø3    +      DB   X
ØØØ3 'Ø4    +      DB   X
ØØØ4 'Ø5    +      DB   X
ØØØ5 'Ø6    +      DB   X
ØØØ6 'Ø7    +      DB   X
ØØØ7 'Ø8    +      DB   X
ØØØ8 'Ø9    +      DB   X
ØØØ9 'ØA    +      DB   X
                   END
```

Undefinierte Wiederholung

```
IRP <dummy>,<parameter, eingeschlossen in spitze Klammern>
  .
  .
  .
ENDM
```

Die Parameter muessen in spitze Klammern eingeschlossen werden!

Die Parameter koennen gueltige Symbole, Zeichenketten, Zahlen oder Zeichenkettenkonstanten sein.
Der Anweisungsblock wird fuer jeden Parameter wiederholt. Jede Wiederholung ersetzt den naechsten Parameter fuer jedes Auftreten von <dummy> in dem Block.
Wenn ein Parameter Null ist (z,B. <>), wird der Block einmal mit einem Nullparameter ausgefuehrt.

Beispiel:

```
        IRP  X,<1,2,3,4,5,6,7,8,9,1Ø>
        DB   X
        ENDM
```

Dieses Beispiel erzeugt denselben Code (DB 1 - DB 10) wie im Beispiel bei REPT.
Wenn IRP innerhalb eines Makrodefinitionsblockes verwendet wird, werden die spitzen Klammern um die Parameter in der Makroaufruf-Anweisung entfernt, bevor die Parameter an den Makroblock uebergeben werden.
Ein Beispiel, das den gleichen Code wie oben erzeugt, verdeutlicht das Entfernen der Klammern von den Parametern:

```
MAC  MACRO     X
     IRP       Y,<X>
     DB        Y
     ENDM
     ENDM
```

Wenn die Makroaufruf-Anweisung

```
        MAC <1,2,3,4,5,6,7,8,9,1Ø>
```

uebersetzt wird, dann sieht die Makroerweiterung folgendermassen aus:

```
        IRP       Y,<1,2,3,4,5,6,7,8,9,1Ø>
        DB        Y
        ENDM
```

Die spitzen Klammern um die Parameter sind entfernt worden und der Inhalt wird als ein einziger Parameter uebergeben.

Undefinierte Wiederholung mit Zeichen

```
IRPC <dummy>,<string>
  .
  .
  .
ENDM
```

Die Anweisungen in dem Block werden fuer jedes Zeichen der Zeichenkette wiederholt.
Jede Wiederholung ersetzt das naechste Zeichen der Zeichenkette fuer jedes Auftreten von <dummy> im Block.

Beispiel:

```
IRPC    X,Ø123456789
DB      X+1
ENDM
```

Dieses Beispiel erzeugt den gleichen Code (DB 1 - DB 1Ø) wie die beiden vorhergehenden Beispiele.

9.4.2.3. Beendigungs-Pseudooperationen

Makro-Ende

```
ENDM
```

ENDM teilt dem Assembler mit, dass der Makro- oder Wiederholungsblock zu Ende ist.

Jede Pseudooperation MACRO, REPT, IRP, IRPC muss mit einem zugehoerigen ENDM beendet werden. Andererseits wird am Ende jedes Passes die Meldung "Unterminated REPT/IRP/IRPC/MACRO" (nicht beendetes REPT/...) erzeugt.

Ein ueberzaehliges ENDM verursacht einen "O"-Fehler.
Wenn ein Makro- oder Wiederholungsblock verlassen werden soll, bevor die Erweiterung vollstaendig ist, wird die Pseudooperation EXITM verwendet.

Verlassen des Makro

```
EXITM
```

EXITM wird innerhalb eines Makro- oder Wiederholungsblockes zum vorzeitigen Beenden einer Erweiterung verwendet, wenn irgendeine Bedingung die weitere Erweiterung unnoetig oder unerwuenscht macht. Meistens wird EXITM in Verbindung mit einer bedingten Pseudooperation verwendet.

Wenn ein EXITM uebersetzt wird, wird die Erweiterung sofort abgebrochen. Die restliche Erweiterung oder uebrige Wiederholungen werden nicht generiert.

Wenn der Block, der das EXITM enthaelt, in einem anderen Block verschachtelt ist, wird die Erweiterung in der aeusseren Ebene fortgesetzt.

Beispiel:

```
MAC   MACRO     X
Y     SET       Ø
      REPT      X
Y     SET       Y + 1
      IFE       Y - ØFFH ; Test Y
      EXITM              ; wenn Ø, verlassen REPT
      ENDIF
      DB        Y
      ENDM
      ENDM
```

9.4.2.4. Pseudooperation fuer Makro-Symbole

LOCAL <dummy>[,<dummy>...]

Die Pseudooperation LOCAL wird nur innerhalb eines Makrodefinitionsblockes verwendet.

Wenn LOCAL ausgefuehrt wird, legt der Makroassembler ein einmaliges Symbol fuer jedes <dummy> an und ersetzt dieses Symbol fuer jedes Auftreten von <dummy> in der Erweiterung.
Diese einmaligen Symbole werden meistens zur Definition einer Marke innerhalb eines Makros verwendet, auf diese Weise vermeidet man Mehrfachdefinitionen bei mehrfacher Erweiterung des Makros.

Die vom Assembler angelegten Symbole reichen von ..ØØØ1 bis ..FFFF. Der Anwender sollte eigene Symbole der Form ..nnnn vermeiden! Die LOCAL-Anweisung muss allen anderen Anweisungstypen in der Makrodefinition vorausgehen.

Beispiel:

```
MAC       MACRO     NUM,Y
          LOCAL     A,B,C,D,E
A:        DB        7
B:        DB        3
C:        DB        Y
D:        DB        Y+1
E:        DW        NUM+1
          JMP       A
          ENDM
          MAC       ØCØØH,ØBEH
          END
```

erzeugt folgende Uebersetzungliste:

```
                    MAC         MACRO       NUM,Y
                                LOCAL       A,B,C,D,E
                    A:          DB          7
                    B:          DB          8
                    C:          DB          Y
                    D:          DB          Y+1
                    E:          DW          NUM+1
                                JMP         A
                                ENDM
                                MAC         ØCØØH,ØBEH
ØØØØ' Ø7    + ..ØØØØ:   DB          7
ØØØ1' Ø8    + ..ØØØ1:   DB          8
ØØØ2' BE    + ..ØØØ2:   DB          ØBEH
ØØØ3' BF    + ..ØØØ3:   DB          ØBEH+1
ØØØ4' ØCØ1  + ..ØØØ4:   DW          ØCØØH+1
ØØØ6'       +           JMP         ..ØØØØ
                         END
```

9.4.2.5. Spezielle Makro-Operatoren

Besondere spezielle Operatoren koennen in einem Makroblock verwendet werden, um zusaetzliche Assemblerfunktionen auszuwaehlen.

& Ampersand verkettet Text oder Symbole. (Das Ampersand kann nicht in einer Makroaufruf-Anweisung verwendet werden.)
Ein symbolischer Parameter wird in der Erweiterung nicht ersetzt, wenn nicht unmittelbar vorher ein & steht.
Ein Symbol wird aus Text und symbolischem Parameter gebildet, indem zwischen beide ein & geschrieben wird.

Beispiel:

```
ERRGEN      MACRO       X
ERROR&X:    PUSH        B
            MVI         B,'&X'
            JMP         ERROR&X
            ENDM
```

Der Aufruf ERRGEN A wird erzeugen:

```
ERROR&A:    PUSH        B
            MVI         B,'A'
            JMP         ERROR&A
```

;; In einem Block wird ein Kommentar, dem 2 Semikolons vorausgehen, nicht als Teil der Erweiterung erhalten (er erscheint bei .LALL nicht auf dem Protokoll).
Ein Kommentar nach nur einem Semikolon wird erhalten und erscheint in der Erweiterung.

! Ein Ausrufezeichen kann in einem Argument geschrieben werden, um anzuzeigen, dass das naechste Zeichen als Literal genommen werden soll.
Deswegen ist !; dasselbe wie <;>.

% Das Prozentzeichen wird nur in einem Makroargument zum Konvertieren des auf das % folgenden Ausdruckes (meistens ein Symbol) in eine Zahl in der aktuellen Zahlenbasis (festgesetzt mit .RADIX) verwendet.
Waehrend der Makroerweiterung ersetzt diese Zahl den symbolischen Parameter.
Die Verwendung des %-Operators erlaubt den Makroaufruf mit einer Zahl.
(Meistens ist ein Makroaufruf ein Aufruf mit einer Referenz, wobei der Text des Makroarguments exakt den symbolischen Parameter ersetzt.)
Der auf % folgende Ausdruck muss den gleichen Regeln genuegen, wie der Ausdruck bei einer DS-Pseudooperation. Das heisst, es ist ein gueltiger Ausdruck erforderlich, der eine absolute Konstante (nichtverschieblich) ergibt.

Beispiel:

```
PRINTE    MACRO     MSG,N
          .PRINTX    *MSG,N*
          ENDM

SYM1      EQU       1ØØ
SYM2      EQU       2ØØ
          PRINTE <SYM1+SYM2=>,%(SYM1+SYM2)
```

Normalerweise wuerde beim Makroaufruf der symbolische Parameter N durch die Kette (SYM1+SYM2) ersetzt.
Das Resultat waere:

```
PRINTX*SYM1+SYM2=,(SYM1+SYM2)*
```

Wenn das % vor dem Parameter steht, wird folgendes erzeugt:

```
PRINTX*SYM1+SYM2=,3ØØ*
```

9.4.3. Pseudooperationen zur bedingten Assemblierung

Die bedingten Pseudooperationen ermoeglichen dem Anwender Codebloecke zu entwerfen, die spezielle Bedingungen testen und entsprechend vorgehen.

Alle Bedingungen haben folgendes Format:

```
IFxxxx [argument]  |        COND [argument]
  .                |          .
  .                |          .
  .                |          .
[ELSE              |        [ELSE
  .                |          .
  .                |          .
  .  ]             |          .  ]
ENDIF              |        ENDC
```

Zu jedem IFxxxx muss ein zugehoeriges **ENDIF** die Bedingung abschliessen.
Zu jedem COND muss ein zugehoeriges **ENDC** die Bedingung abschliessen. Andererseits wird die Meldung "Unterminated Conditional" (nicht beendete Bedingung) am Ende jedes Passes erzeugt.
Ein ENDIF ohne zugehoeriges IF oder ein ENDC ohne zugehoeriges COND verursacht einen "C"-Fehler.

Der Assembler bewertet eine Bedingungsanweisung mit "wahr" (ist gleich FFFFH oder -1 oder jeder Wert ungleich Null) oder mit "falsch" (ist gleich Null).

Der Code in dem Bedingungsblock wird uebersetzt, wenn der Wert der Bedingung entspricht, die in der Bedingungsanweisung definiert wurde. Wenn der Wert dem nicht entspricht, ignoriert der Assembler den Bedingungsblock entweder vollstaendig oder, wenn er eine optionale ELSE-Anweisung enthaelt, uebersetzt er nur den ELSE-Teil.

Bedingungen koennen bis zu 255mal geschachtelt werden. Jedes Argument einer Bedingung muss im Pass 1 bekannt sein, um einen "V"-Fehler und unkorrekte Auswertung zu vermeiden. Der Ausdruck fuer IF/IFT/COND und IFF/IFE muss einen Wert haben, der vorher definiert wurde und der absolut ist.

Wenn der Name nach einem IFDEF oder IFNDEF definiert wurde, betrachtet der Assembler ihn im Pass 1 als undefiniert; aber er wird im Pass 2 definiert.

Jeder Bedingungsblock kann optional die Pseudooperation ELSE enthalten, die die Moeglichkeit gibt, alternativen Code zu erzwingen, wenn die entgegengesetzte Bedingung auftritt.
Fuer ein IFxxxx/COND ist nur ein ELSE erlaubt.
Das ELSE ist immer mit dem zuletzt eroeffneten IF verbunden.
Eine Bedingung mit mehr als einem ELSE oder ein ELSE ohne eine Bedingung wird einen "C"-Fehler verursachen.

Bedingte Pseudooperation

```
IF <exp>
IFT <exp>
COND <exp>
```

Die Anweisungen innerhalb des Bedingungsblockes werden uebersetzt, wenn <exp> nicht Null ergibt.

IFE <exp>
IFF <exp>

Die Anweisungen innerhalb des Bedingungsblockes werden uebersetzt, wenn <exp> den Wert Null hat.

IF1 Pass 1 Bedingung

Die Anweisungen innerhalb des Bedingungsblockes werden uebersetzt, wenn sich der Assembler im Pass 1 befindet.

IF2 Pass 2 Bedingung

Die Anweisungen innerhalb des Bedingungsblockes werden uebersetzt, wenn sich der Assembler im Pass 2 befindet.

IFDEF <symbol>

Die Anweisungen innerhalb des Bedingungsblockes werden uebersetzt, wenn das Symbol definiert ist oder als EXTERNAL erklaert wurde.

IFNDEF <symbol>

Die Anweisungen innerhalb des Bedingungsblockes werden uebersetzt, wenn das Symbol nicht definiert und nicht als EXTERNAL erklaert wurde.

IFB <arg>

Die spitzen Klammern um <arg> sind erforderlich. Die Anweisungen im Bedingungsblock werden uebersetzt, wenn das Argument leer (nicht angegeben) oder Null (<>) ist.

IFNB <arg>

Die spitzen Klammern um das <arg> sind erforderlich. Die Anweisungen innerhalb des Bedingungsblockes werden uebersetzt, wenn das Argument nicht leer ist.
Dies wird verwendet, um symbolische Parameter zu testen.

IFIDN <arg1>,<arg2>

Die spitzen Klammern um <arg1> und <arg2> sind erforderlich.
Die Anweisungen innerhalb des Bedingungsblockes werden uebersetzt, wenn die Kette <arg1> identisch der Kette <arg2> ist.

IFDIF <arg1>,<arg2>

Die spitzen Klamern um <arg1> und <arg2> sind erforderlich.
Die Anweisungen innerhalb des Bedingungsblockes werden uebersetzt, wenn die Kette <arg1> verschieden von der Kette <arg2>

ist.

ELSE

ELSE bietet die Moeglichkeit, alternativen Code zu erzeugen, wenn die entgegengesetzte Bedingung auftritt. ELSE kann mit jeder bedingten Pseudooperation verwendet werden.

ENDIF
ENDC

Diese Pseudooperationen schliessen den Bedingungsblock ab. Sie muessen zu jeder verwendeten bedingten Pseudooperation geschrieben werden.
ENDIF muss ein zugehoeriges IFxxxx haben und ENDC ein zugehoeriges COND.

9.5. Abarbeitung des Assemblers

Wenn das Assemblerprogramm vollstaendig entworfen ist, kann es uebersetzt werden.
Der Assembler uebersetzt die Quelldateianweisungen einschliesslich der Erweiterung von Makros und Wiederholungspseudooperationen.

Das Ergebnis ist ein verschieblicher Objektcode, der mit dem Programmverbinder gebunden und geladen werden kann. Der verschiebliche Objektcode kann in eine Diskettendatei, die vom Assembler den Dateityp .REL bekommt, geschrieben werden. Die uebersetzte Datei (.REL) ist nicht ausfuehrbar.
Die Datei wird erst nach der Behandlung durch den Programmverbinder ausfuehrbar.

Der Assembler benoetigt ungefaehr 19K Speicherplatz und hat eine Abarbeitungsgeschwindigkeit von mehr als 1ØØØ Zeichen je Minute.
Der Assembler arbeitet unter dem Betriebssystem SCPX 1526.

Der Assembler uebersetzt die Quelldatei in zwei Paessen.
Waehrend des 1. Passes wertet er die Programmanweisungen aus, berechnet, wieviel Code er erzeugen wird, bildet eine Symboltabelle, in der jedem Symbol ein Wert zugewiesen wird und erweitert die Makroaufruf-Anweisungen.

Waehrend des 2. Passes setzt er in die Symbole und Ausdruecke die Werte aus der Symboltabelle ein, erweitert erneut die Makroaufruf-Anweisungen und gibt den verschieblichen Code aus. Der Assembler prueft die Werte der Symbole, Ausdruecke und Makros waehrend beider Paesse. Er gibt einen Phasenfehlercode zurueok, wenn der Wert waehrend des 2. Passes nicht mit dem Wert waehrend des 1. Passes uebereinstimmt.

Bevor mit dem Assembler gearbeitet werden kann, muss die Diskette mit dem Programm ASM.COM in das zu verwendende Laufwerk eingelegt werden.
Die Diskette, auf der sich die Quelldatei befindet, muss ebenfalls in einem Laufwerk liegen.

9.5.1 Aufrufen des Assemblers

Der Assembler wird aufgerufen durch die Eingabe von:

```
ASM
```

Die Programmdatei ASM.COM wird geladen. Der Assembler gibt einen Stern (*) aus und zeigt damit an, dass er bereit ist, eine Kommandozeile entgegenzunehmen.

9.5.2. Die Kommandozeile des Assemblers

Die Kommandozeile des Assemblers setzt sich aus 4 Feldern zusammen:

```
[Object][,[List]]=Source[/Switch]
```

Die Kommandozeile kann als eigene Zeile geschrieben werden oder auf dieselbe Zeile wie das ASM-Kommando.
(Wenn Kommandozeile und ASM-Kommando auf derselben Zeile stehen, kehrt der Assembler nicht mit * zurueck, sondern gibt die Steuerung nach Beenden an das Betriebssystem.)
Bei getrennter Eingabe von ASM und Kommandozeile besteht zusaetzlich die Moeglichkeit, eine weitere Uebersetzung auszufuehren, ohne dass der Assembler wieder aufgerufen wird. Wenn die Uebersetzung beendet ist, meldet sich der Assembler dann sofort mit einem * und wartet auf eine neue Kommandozeile. Der Assembler wird in diesem Fall durch Druecken von ^C beendet.

Wenn nur ein Programm uebersetzt werden soll, ist es bequemer, das ASM-Kommando und die Kommandozeile auf dieselbe Zeile zu schreiben; es erfordert weniger Eingaben und bietet die Moeglichkeit, die Assembler-Operation einem SUBM-Kommando mitzuteilen.
Wenn das ASM-Kommando und die Kommandozeile zusammenstehen, kehrt der Assembler automatisch nach Abschluss zum Betriebssystem zurueck.

Bemerkung:

Wenn die Kommandozeile eine vom ASM-Kommando separate Zeile ist, kann die Kommandozeile nur in Grossbuchstaben eingegeben werden! Wird anders gearbeitet, erscheint die Meldung "?Command Error". Wenn das ASM-Kommando und die Kommandozeile eine Zeile bilden, koennen die Eingaben in grossen oder kleinen Buchstaben oder gemischt gemacht werden. Das Betriebssystem

wandelt vor der Uebergabe alle Eingaben in grosse Buchstaben .

9.5.2.1. Source (=dateiname)

Um ein Quellprogramm zu uebersetzen, muss wenigstens ein Gleichheitszeichen (=) und der Quelldateiname eingegeben werden.

Das "=dateiname" zeigt an, welche Quelldatei zu uebersetzen ist. Wenn die Diskette mit der Quelldatei nicht im aktuell zugewiesenen Laufwerk liegt, muss die Laufwerksauswahl als Teil von dateiname mit spezifiziert werden.
Enthaelt der Dateiname keinen Dateityp, nimmt der Assembler an, dass er .MAC ist. Ist er nicht .MAC, muss er als Teil von dateiname unbedingt angegeben werden.

Ueber andere Moeglichkeiten der Zuweisung von Geraet und Dateityp siehe Pkt. 9.5.2.6.

Die Eintragung source ist die einzige, die fuer ASM unbedingt erforderlich ist.
Das einfachste Kommando lautet:

```
ASM =source
```

Dieses Kommando veranlasst den Assembler, die Quelldatei zu uebersetzen und das Ergebnis in einer verschieblichen Objectcodedatei (genannt .REL-datei) mit demselben Namen wie die Quelldatei abzulegen.

Beispiel:

Die Quelldatei soll PROG.MAC heissen, dann lautet die Kommandozeile

```
ASM =PROG
```

und es wird eine uebersetzte Datei mit dem Namen

```
PROG.REL
```

erzeugt.

Als zusaetzliche Option kann nur ein Komma (,) auf der linken Seite des Gleichheitszeichens geschrieben werden. Dann werden vom Assembler alle Ausgabedateien (object und list) unterdrueckt.

Die Kommandozeile

```
ASM ,=PROG
```

veranlasst den Assembler, die Datei PROG.MAC zu uebersetzen, aber die erzeugten Dateien nicht auszugeben.

Die Programmierer verwenden diese Kommandozeile, um die Syntax im Quellprogramm zu testen, bevor das Uebersetzungsergebnis ausgegeben wird. Weil keine Dateien ausgegeben werden, ist der Assembler rascher beendet, die Fehler eher bekannt.

9.5.2.2. Object (dateiname)

Die Eintragung object ist immer optional.
Bestimmte Bedingungen zwingen dazu, object anzugeben.

Die Objectdatei schreibt das uebersetzte Programm in eine Diskettendatei. Diese wird vom Programmverbinder zum Erzeugen eines ausfuehrbaren Programms verwendet.

Wenn beide Eintragungen, object und list, in der Kommandozeile weggelassen sind (wie bei =source), erzeugt der Assembler eine Objectdatei mit dem gleichen Dateinamen wie die Quelldatei, aber mit der Standarderweiterung .REL.

Wenn ein vom Quelldateinamen verschiedener Name gewuenscht wird, muss dieser dateiname im Object-Feld eingetragen werden. Der Assembler haengt dann den Dateityp .REL an, wenn kein Dateityp eingetragen war.
Wenn beide Dateien - Objectcode und Druckdatei - gewuenscht werden, muessen beide Felder - object und list - ausgefuellt werden. Es wird nur eine Druckdatei erzeugt, wenn das Object-Feld frei bleibt.
Diese Form verwendet der Programmierer zum Testen der Syntax. Der Programmausdruck ist ausserdem eine Testhilfe.

Der Name fuer die Objektcodedatei kann jeder gueltige Dateiname sein. Es ist moeglich, einen anderen als den Quelldateinamen zu waehlen, aber es hat sich als guenstig erwiesen, alle Dateien, die zu einem Programm gehoeren, mit dem gleichen Namen zu versehen.

9.5.2.3. List (,dateiname)

Die Eintragung list ist immer optional. Das Komma vor der list-Eintragung muss geschrieben werden.
Wird eine Druckdatei gewuenscht, muss ein Dateiname in das list-Feld geschrieben werden (Eine Alternative zu dieser Regel siehe Pkt. 9.5.2.4.).

Der Assembler haengt .PRN als Standarddateityp an, wenn keine andere in der list-Eintragung angegeben wurde.

Die Kommandozeile

```
ASM ,PROG=PROG
```

uebersetzt die Datei PROG.MAC (Quelldatei) und erzeugt die

Druckdatei PROG.PRN. Eine Objectdatei wird nicht erzeugt.

Die Druckdatei kann denselben Namen wie die Quelldatei haben oder jeden anderen gueltigen Dateinamen.

Wenn auf der linken Seite des Gleichheitszeichens nur ein Komma und keine Dateinamen stehen, uebersetzt der Assembler die Quelldatei, aber er erzeugt keine Ausgabedateien.

Dieses Kommando

```
ASM ,=PROG
```

erlaubt die Pruefung der Quelldatei auf Syntaxfehler, bevor das uebersetzte Programm auf die Diskette geschrieben wird. Waehrend der Assembler die Pruefung der Fehler immer durchfuehrt, werden mit diesem Kommando keine Ausgabedateien erzeugt und damit ist der Assemblerlauf um einiges schneller.

Wenn die Uebersetzung beendet ist, gibt der Assembler folgende Meldung aus:

```
[xx][No] Fatal Errors [,xx warnings]
```

Diese Meldung zeigt die Anzahl der schwerwiegenden Fehler und Warnungen, die im Programm aufgetreten sind, an.
Diese Meldung wird am Ende des Uebersetzungslaufes auf den Bildschirm und auf die Druckdatei ausgegeben.
Wenn diese Meldung erscheint, ist der Assembler fertig.
Heisst die Meldung "No Fatal Errors", dann ist die Uebersetzung vollstaendig und erfolgreich.

Beispiele:

ASM =TEST	Uebersetzen der Quellcodedatei TEST.MAC und Erzeugen der Objektcodedatei TEST.REL
ASM ,=TEST	Uebersetzen der Quellcodedatei TEST.MAC, ohne dass Ausgabedateien erzeugt werden.
ASM TEST,TEST=TEST	Uebersetzen der Quellcodedatei TEST.MAC, Erzeugen der Objectcode-Datei TEST.REL und der Druckdatei TEST.PRN
ASM OBJECT=TEST	Uebersetzen der Quellcodedatei und Erzeugen der Objektcodedatei OBJECT.REL.

9.5.2.4. Schalter (/switch)

Das Assembler-Kommando kann neben dem Uebersetzen und Erzeugen von Object- und Druckdatei einige zusaetzliche Funktionen ausfuehren.

Diese zusaetzlichen Kommandos werden an das Ende der Kommandozeile eingegeben.
Eine solche Eingabe veranlasst den Assembler eine zusaetzliche oder alternative Funktion "einzuschalten": Diese Eingaben werden Schalter genannt.
Schalter sind Buchstaben, vor denen ein Schraegstrich (/) steht. Es kann jede beliebige Anzahl Schalter eingegeben werden, aber vor jedem Schalter muss ein Schraegstrich stehen.

Beispiel:

```
ASM ,=PROG/L/R
```

Folgende Schalter des Assemblers sind moeglich:

Schalter	Bedeutung
/O	Der Assembler druckt die Adressen in der Druckdatei oktal (zur Basis 8).
/P	Jedes /P legt einen extra Stack von 256 Bytes zur Verwendung waehrend der Uebersetzung an. /P sollte dann verwendet werden, wenn waehrend der Uebersetzung ein Stack-Ueberlauf-Fehler aufgetreten ist. Sonst ist er nicht notwendig.

/R	Erzwingt das Erzeugen einer Objektdatei mit dem gleichen Namen wie die Quelldatei. Das kann anstelle der Angabe des Dateinamens im Objekt-Feld der Kommandozeile verwendet werden. Die Angabe dieses Schalters ist geeignet, wenn eine .REL-datei erzeugt werden soll, aber vergessen wurde, einen Dateinamen in das Object-Feld einzutragen und ein Komma und ein Druckdateiname oder nur ein Komma vor dem Gleichheitszeichen geschrieben wurde. Also wenn geschrieben wurde:
	
	`ASM ,PROG=PROG`
	
	oder
	
	`ASM ,=PROG`
	
	dann kann auch eine .REL-Datei erzwungen werden durch einfaches Anfuegen eines /R vor dem Druekken von <ENTER>. Die Kommandozeile heisst dann:
	
	`ASM ,PROG=PROG/R`
	
	oder
	
	`ASM ,=PROG/R`

Schalter	Bedeutung
/X	Der /X-Schalter setzt den Standard und die aktuelle Zuordnung auf Unterdruecken der Liste falscher Bedingungen. Nichtvorhandensein von /X bedeutet, der Standard und die aktuelle Zuordnung werden auf Drucken der falschen Bedingungen gesetzt. /X wird oft in Verbindung mit der bedingten Pseudooperation .TFCOND verwendet (Siehe Pkt. 9.4.1.5.).
/L	Erzwingt das Erzeugen einer Druckdatei mit dem gleichen Namen wie die Quelldatei. Das kann verwendet werden, statt der Angabe eines Druckdateinamens im list-Feld der Kommandozeile. Wenn eingegeben wurde: ASM =PROG oder ASM ,=PROG oder ASM PROG=PROG dann kann durch einfaches Anhaengen eines /L vor dem Druecken von <ENTER> eine Druckdatei erzwungen werden. Die Kommandozeile heisst dann: ASM =PROG/L oder ASM ,=PROG/L oder ASM PROG=PROG/L
/C	Veranlasst den Assembler, eine spezielle Druckdatei mit dem gleichen Namen wie die Quelldatei anzulegen, fuer die Verwendung mit REF. Wenn das Erzeugen einer Cross-Referenz gewuenscht wird, muss das Programm mit diesem Schalter uebersetzt werden (siehe Pkt. 9.7.).
/Z	Veranlasst den Assembler, Z80-Operationscodes zu uebersetzen. Dieser Schalter muss angegeben werden, wenn die Quelldatei im Z80-Code geschrieben wurde, aber keine .Z80-Pseudoopration enthaelt.
/I	Veranlasst den Assembler, 8080-Operationscodes zu uebersetzen. Dieser Schalter muss angegeben werden, wenn die Quelldatei im 8080-Code geschrieben wurde, aber keine .8080-Pseudooperation enthaelt.

Schalter	Bedeutung
/H	Das ist der Standard. Der Assembler druckt die Adressen in der Druckdatei hexadezimal.
/M	Der /M-Schalter initialisiert einen Blockdatenbereich, der mit DS definiert wurde auf ØØH. Andererseits wird der Bereich nicht initialisiert. Das heisst, DS initialisiert den Speicherbereich nicht automatisch auf ØØH. In diesem Fall kann man nicht wissen, was in diesem Speicherbereich steht.

9.5.2.5. Zusaetzliche Angaben der Kommandozeile

In jedes Feld der Kommandozeile koennen noch zwei zusaetzliche Arten von Eintragungen vorgenommen werden - der Dateityp und die Geraetezuordnung. Diese beiden Arten sind aktuelle Teile einer "Dateispezifikation".
Eine Dateispezifikation besteht aus dem Geraet, wo die Datei lokalisiert ist, dem Namen der Datei und dem Dateityp.

Meist wird die Standardzuweisung fuer das Geraet und der Dateityp verwendet; sie wird dann vom Assembler eingefuegt, wenn diese Positionen in der Kommandozeile nicht ausgefuellt waren.
Die Standardannahmen hindern nicht daran, entweder Dateityp oder Geraetezuweisung einzugeben, einschliesslich der Eingaben, die den Standard darstellen.
Der Programmierer kann diese zusaetzlichen Eingaben in jeder Kombination angeben oder weglassen.

Das Format fuer eine Dateispezifikation ist:

```
     dev:dateiname.ext
wobei
     dev:        1-3 Buchstaben Geraetezuweisung (zwingend) gefolgt von einem Doppelpunkt

     dateiname:1-8 Zeichen Dateiname

     .ext:       1-3 Zeichen Dateityp, dem (zwingend) ein Punkt vorausgehen muss.
```

Dateityp (.ext)

Zum Unterscheiden zwischen Quelldatei, Objectdatei und Druckdatei fuegt der Assembler einen Dateityp an den Dateinamen an. Der Dateityp ist eine aus 3 Zeichen bestehende Mnemonik, die an den Dateinamen angehaengt wird, wobei zwischen Dateinamen und Dateityp ein Punkt (.) steht.

Der vom Assembler ergaenzte Dateityp wird Standarddateityp genannt.

Die Standarddateitypen sind:

.REL Objectdatei
.PRN Druckdatei
.COM absolute (ausfuehrbare) Datei

Wenn fuer die Quelldatei kein Dateityp angegeben ist, nimmt der Assembler an, dass der Dateityp .MAC ist.

Es kann aber auch ein eigener Dateityp angegeben werden, wenn dies notwendig oder wuenschenswert ist. Es ist ein Nachteil, dass beim Rufen der Datei dann immer der Dateityp mit eingegeben werden muss. Bei Verwendung des Standarddateityps kann beim Ruf der Datei diese Angabe entfallen.

Geraetezuweisung (dev:)

Jedes Feld in der Kommandozeile kann auch eine Geraetezuweisung beinhalten.

Die im source-Feld spezifizierte Geraetezuweisung teilt dem Assembler mit, wo die Quelldatei zu finden ist. Die im Object-Feld oder list-Feld spezifizierte Geraetezuweisung sagt dem Assembler, wohin die Ausgabe der Objectdatei bzw. der Druckdatei erfolgen soll.
Wenn in einem der Felder die Geratezuweisung weggelassen ist, nimmt der Assembler standardmaessig das aktuell zugewiesene Laufwerk an.
Das heisst, wenn das Geraet das aktuell zugewiesene Laufwerk ist, braucht die Geraetezuweisung nicht spezifiziert zu werden. Es ist notwendig, das Geraet zu spezifizieren, wenn ein anderes als das aktuell zugewiesene Laufwerk waehrend der Uebersetzung verwendet werden soll.

Beispiel:

Die Diskette mit dem Assembler liegt im Laufwerk A, die Programmdiskette im Laufwerk B. Die .REL-Datei soll ebenfalls auf B ausgegeben werden.
Dann muss in der Kommandozeile nur

```
ASM =B:PROG
```

angegeben werden.

Wenn die .REL-Datei ausgegeben ist, ist das zugewiesene Laufwerk B. (Jedoch, wenn der Assembler beendet ist, ist A wieder das zugewiesene Laufwerk.)

Im Gegensatz dazu das folgende Beispiel:

Beispiel:

Die Quelldatei befindet sich mit auf der Diskette, auf der der Assembler ist. Diese Diskette liegt im Laufwerk A. Die .REL-Datei soll auf die im Laufwerk B liegende Diskette ausgegeben werden.
Die Kommandozeile lautet dann:

```
ASM B:=A:PROG
```

Man sollte es sich zur Regel machen:

Wenn man nicht sicher ist, ob eine Geraetezuweisung angegeben werden muss, dann sollte man sie auf jeden Fall angeben. Das ist der sichere Weg, jede Datei auf den richtigen Platz zu bringen!

Gueltige Geraetezuweisungen fuer den Assembler sind:

A:,B:,C:,...	Diskettenlaufwerke
LST:	Drucker
TTY:	Bildschirm oder Tastatur

Geraetezuweisung als Dateiname

Eine weitere Option ist die Angabe der Geraetezuweisung ohne Dateiname in der Kommandozeile an die Stelle des Dateinames. Mit der Verwendung dieser Option werden verschiedene Resultate erzielt, abhaengig davon, welches Geraet in welches Feld eingetragen wird.

Beispiel:

```
ASM ,TTY:=TTY:
```

ermoeglicht, eine unmittelbar eingegebene Zeile zu uebersetzen und auf Syntax- oder andere Fehler zu pruefen.

Aehnliches bewirkt:

```
ASM ,LST:=TTY:
```

jedoch wird das Ergebnis nicht auf den Bildschirm, sondern auf den Drucker ausgegeben.

Wenn eines der Kommandos (,TTY:=TTY: oder ,LIST:=TTY:) verwendet wird, sollte als erstes eine END-Anweisung eingegeben werden. Der Assembler muss in den Pass 2 versetzt werden, bevor er Code ausgibt. Wenn einfach begonnen wird, Anweisungszeilen einzugeben ohne vorheriges END, wird keine Antwort kommen, bis eine END-Anweisung auftritt. Danach muessen alle Anweisungen erneut eingegeben werden, ehe der generierte Code sichtbar wird.

Tabelle 1 verdeutlicht das. Diese Tabelle soll einige Moeglichkeiten zeigen, sie ist keine vollstaendige Liste aller Kombinationen.

Tabelle 1: Wirkungen der Geraetezuweisung ohne Dateiname

dev:	object	,list	=source
A:,B:, C:,D:,	Schreiben der Datei auf das spezifizierte Laufwerk	Schreiben der Datei auf das spezifizierte Laufwerk	nicht moeglich (ein Dateiname muss spezifiziert werden)
LST:	nicht moeglich (unlesbares Dateiformat)	Schreiben der Liste auf den Drucker	nicht moeglich (nur Ausgabe)
TTY:	nicht moeglich (unlesbares Dateiformat)	"Schreiben" der Liste auf den Bildschirm	"Lesen" des Quellprogrammes von der Tastatur

9.5.3. Format des Assemblerprotokolls

Die Druckliste des Assemblers besteht aus 2 Teilen in 2 verschiedenen Formaten, zum einen die Zeilen der Assemblerliste und zum anderen die Symboltabelle.

9.5.3.1. Format der Assemblerliste

Jede Seite der Assemblerliste beginnt mit 2 Kopfzeilen. Wenn die Quelldatei keine Kopfzeilen aufweist (weder TITLE noch SUBTTL wurden angegeben), sind diese Teile der Kopfzeilen leer.

Das Format ist:

```
[TITLE text]          ASM(SCPX) 152Ø V n.n   X
[SUBTTL text]
```

Dabei sind:

TITLE text — ist der Text, der mittels der Pseudooperation TITLE in der Quelldatei angegeben wurde. War in der Quelldatei keine TITLE-Pseudooperation vorhanden, dann ist dieser Platz in der Kopfzeile leer.

n.n — ist die Versionsnummer des Assemblers

X — ist die Seitennummer.

Diese wird nur angezeigt und erhoeht, wenn in der Quelldatei die Pseudooperation .PAGE auftritt, oder wenn die gerade bearbeitete Seite gefuellt ist.

SUBTTL text ist der Text, der mitels der Pseudooperation SUBTTL in der Quelldatei angegeben wurde.
War in der Quelldatei keine SUBTTL-Pseudooperation vorhanden, dann ist dieser Platz in der Kopfzeile leer.

Den beiden Kopfzeilen folgt eine Leerzeile.
In der darauf folgenden Zeile beginnt der Text der Druckdatei.

Das Format einer Druckzeile ist:

[error] ####m xx xxxxm [w] text

Dabei sind:

error stellt einen Fehlercode aus einem Zeichen bestehend dar.
Ein Fehlercode wird nur gedruckt, wenn in dieser Zeile ein Fehler aufgetreten ist. Sonst bleibt der Platz frei.

stellt den Zuordnungszaehler dar.
Die Zahl ist eine 4-stellige Hexadezimalzahl oder eine 6-stellige Oktalzahl.
Die Basis des Zuordnungszaehlers wird durch die Verwendung des /O- oder /H-Schalters im switch-Feld der Kommandozeile des Assemblers bestimmt.
Der Standard ist hexadezimal.

m zeigt den Zuordnungszaehlermodus an.
Die moeglichen Symbole dafuer sind:

'	Coderelativ
"	Datenrelativ
!	Commonrelativ
<leer>	Absolut
*	External

xx und xxxx stellen den uebersetzten Code dar.
xx ist ein 1-Byte-Wert. 1-Byte-Werten folgt immer ein Leerzeichen.
xxxx ist ein 2-Byte-Wert mit dem hoeherlogische Operatoren.
Operatoren, die wahre oder falsche Bedingungen melden, geben wahr zurueck, wenn das Ergebnis ungleich Ø ist und falsch, wenn das Ergebnis gleich Ø ist.
Die folgenden arithmetischen und logischen Operatoren sind in Ausdruecken erlaubt:

Operator	Definition
NUL	- Gibt wahr zurueck, wenn das Argument (ein Parameter) null ist. Der Rest der Zeile nach NUL wird als Argument zu NUL genommen. Die Bedingung IF NUL <argument> ist falsch, wenn das erste Zeichen des Arguments alles andere als ein Semikolon oder ein <ENTER> ist. Bemerkung: IFB und IFNB fuehren dieselbe Funktion aus, aber sie sind einfacher zu verwenden (siehe Pkt. 9.4.1.5.).
TYPE	- Der TYPE-Operator gibt ein Byte zurueck, das ndern erscheint, wie er in der Quelldatei erfasst wurde.

9.5.3.2. Format der Symboltabelle

Die Seiten der Symboltabelle haben die gleichen Kopfzeilen wie die Assemblerliste.Aber anstelle der Seitennummer erscheint S.

Danach werden alle Makronamen, die im Programm verwendet werden in alphabetischer Reihenfolge aufgelistet.
Im Anschluss daran erscheinen alle Symbole, ebenfalls in alphabetischer Reihenfolge.
Hinter jedem Symbol steht sein Wert, gefolgt von einem der nachfolgenden Zeichen:

I	Public-Symbol
U	Undefiniertes Symbol
C	Commonblock-Name.
	Der Wert, der hierfuer angzeigt wird, ist die Laenge des Blockes in Byte auf hexadezimaler oder oktalen Basis.
*	EXTERNAL
'	Coderelativer Wert
"	Datenrelativer Wert
!	Commonrelativer Wert

9.5.4. Fehlercodes und Fehlermeldungen

Waehrend der Uebersetzung auftretende Fehler veranlassen den Assembler, entweder einen Fehlercode oder eine Meldung zurueckzugeben. Fehlercodes werden durch ein Zeichen dargestellt und in der Spalte 1 der Druckdatei gedruckt.

Auch wenn die Druckdatei nicht auf den Bildschirm ausgegeben wird, erscheinen die fehlerhaften Zeilen trotzdem auf dem Bildschirm.
Fehlermeldungen werden an das Ende der Druckdatei gedruckt oder am Ende nach den Fehlerzeilen auf dem Bildschirm angezeigt.

9.5.4.1. Fehlercodes

Fehlercode	Bedeutung
A	**Argumentfehler** Das zu der Pseudooperation gehoerige Argument befindet sich nicht im korrekten Format oder es liegt ausserhalb des zugelassenen Bereiches.
C	**Fehler der Bedingungsschachtelung** ELSE ohne IF, ENDIF ohne IF, zweimal ELSE fuer ein IF, ENDC ohne COND.
D	**Doppelt definiertes Symbol** Bezugnahme auf ein Symbol, das mehrfach definiert wurde.
E	**Externer Fehler** Verwendung eines externen Symbols, das in diesem Zusammenhang nicht erlaubt ist. (Bsp.: MARKE1 SET NAME##)
M	**Mehrfach definiertes Symbol** Definition eines Symbols, das bereits definiert wurde.
N	**Fehler einer Zahl** Fehler in einer Zahl; meist ein falsches Zeichen, (bsp.: 8Q).
O	**Falscher Operationscode oder nicht einwandfreie Syntax** SET, EQU oder MACRO ohne einen Namen; falsche Syntax in einem Operationscode oder falsche Syntax in einem Ausdruck (nichtpaarige runde Klammern, Anfuehrungszeichen, aufeinanderfolgende Operatoren usw.)
P	**Phasenfehler** Der Wert einer Marke oder eines EQU-Namens ist im Pass 2 anders als im Pass 1.
Q	**Fragwuerdig** Meistens ist eine Zeile nicht ordentlich abgeschlossen (bsp.: MOV AX,BX). Dies ist eine Warnung.

R	**Verschiebung** Nichterlaubte Verwendung einer Verschiebung in einem Ausdruck, (bsp.: abs-rel). Daten, Code und Common-Bereiche sind verschieblich.
Fehlercode	Bedeutung
U	**Undefiniertes Symbol** Ein Symbol, auf das in einem anderen Ausdruck Bezug genommen wird, ist nicht definiert. Fuer einige Pseudooperationen wird im Pass 1 ein V-Fehler erzeugt und im Pass 2 ein U-Fehler (Siehe auch V-Fehler).
V	**Wertfehler** Eine Pseudooperation (bsp.: .RADIX, .PAGE, DS, IF, IFE), hat im Pass 1 einen undefinierten Wert, obwohl er im Pass 1 bekannt sein muesste. Wenn das Symbol spaeter im Program definiert wird, erzeugt der Assembler im Pass 2 keinen U-Fehler.

9.5.4.2. Meldungen

%NO END statement

Keine END-Anweisung.
Entweder sie fehlt oder sie wurde nicht durchlaufen infolge einer falschen Bedingung, eines nichtabgeschlossenen IRP-, IRPC-, REPT-Blockes oder eines abgeschlossenen Makros.

Unterminated conditional

Wenigstens eine Bedingung ist am Ende der Datei nicht abgeschlossen worden.

Unterminated REPT/IRP/IRPC/MACRO

Es ist wenigstens ein Block nicht abgeschlossen worden.

Symbol table full

Bei der Bildung der Symboltabelle durch den Assembler ist der verfuegbare Speicher erschoepft. Die haeufigste Ursache ist eine ganze Anzahl Makrobloecke, die Anweisungen fuer viele Anweisungszeilen enthalten.

Die Makrobloecke werden vollstaendig in der Symboltabelle abgespeichert, einschliesslich der Kommentare, die den Zeilen angehaengt sind innerhalb eines Makroblockes.

Es sollten also alle Makrobloecke im Quellprogramm geprueft werden.
Um die Kommentare innerhalb der Makrobloecke aus der Symboltabelle auszuschliessen, sollten vor die Kommentare zwei Semikolons (;;) geschrieben werden.

Dadurch sollte ausreichend Platz fuer die Uebersetzung des Programms freigeworden sein.

{xx,No} Fatal Errors [,xx warnings]

Anzahl der schweren Fehler und Warnungen, die im Programm aufgetreten sind.

Diese Meldung wird am Ende jeder Uebersetzung auf den Bildschirm und in die Druckdatei ausgegeben.

Wenn diese Meldung erscheint, ist der Assemblerlauf beendet.
Die Meldung "No Fatal Errors" zeigt an, dass die Uebersetzung vollstaendig und erfolgreich ist.

9.6. Beschreibung der Befehle

In diesem Kapitel wird die Wirkung der einzelnen CPU-Befehle beschrieben.
Der Abschnitt 9.6.16. enthaehlt das Abkuerzungsverzeichnis. Im Abschnitt 9.6.17. ist die Arbeit mit den Bedingungsbits (Flags) beschrieben.

9.6.1. Ladebefehle

Die Ladebefehle transportieren Daten intern zwischen den CPU-Registern oder zwischen CPU-Registern und dem Schreib- / Lesespeicher (RAM). Die Befehle muessen eine Ausgangsadresse, von der die Daten zu entnehmen sind, und eine Zieladresse enthalten.
Der Quellspeicherplatz wird durch den Ladebefehl nicht veraendert.

9.6.1.1. 8-Bit-Ladebefehle

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
MOV r,r'	LD r,r'	Inhalt des Registers r' wird in das Register r umgespeichert.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
MVI r,n	LD r,n	Der Direktoperand n wird in das Register r geladen.
MOV r,M	LD r,M	Inhalt des durch Registerpaar HL addressierten Speicherplatzes M wird in das Register r geladen. Es ist auch die Schreibweise LD r,(HL) zulaessig. Register L beinhaltet dabei die niederwertigen 8 Bit der Adresse und das Register H die hoeherwertigen 8 Bit
---	LD r,(IX+d)	Inhalt des durch Register IX plus Verschiebung adressierten Speicherplatzes wird in das Register r geladen.
---	LD r,(IY+d)	Inhalt des durch Register IY plus Veschiebung adressierten Speicherplatzes wird in das Register r geladen.
MOV M,r	LD M,r	Bringt ein Byte aus dem Register auf den Speicherplatz M, dessen Adresse durch den Inhalt des Registerpaares HL spezifiziert wurde. Register L beinhaltet dabei die niederwertigen 8 Bit der Adresse, Register H die hoeherwertigen 8 Bit.
---	LD (IX+d),r	Dieser Befehl bringt Daten aus dem Register r an einen Speicherplatz, dessen Adresse durch den Inhalt des IY-Registers plus Verschiebung spezifiziert ist.
---	LD (IY+d),r	Dieser Befehl bewirkt den Transport von Daten aus dem Register r an einen Speicherplatz, dessen Adresse durch den Inhalt des IY-Registers plus Adressverschiebung spezifiziert ist.
MVI M,n	LD M,n	Dieser Befehl bewirkt den Transport des mit n definierten Direktoperanden an einen Spei-

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
		cherplatz M, dessen Adresse durch den Inhalt des Registerpaares HL spezifiziert ist. Register L beinhaltet dabei die niederwertigen 8 Bit der Adresse und das Register H die hoeherwertigen 8 Bit. Die Schreibweise LD (HL),n ist ebenfalls moeglich.
---	LD (IX+d),n	Dieser Befehl bewirkt den Transport des mit n definierten Direktoperanden an einen Speicherplatz, dessen Adresse durch den Inhalt des IX-Registers plus Adressverschiebung spezifiziert ist.
---	LD (IY+d),n	Ein mit n definierter Direktoperand wird an einen Speicherplatz transportiert, dessen Adresse durch den Inhalt des IY-Registers plus Adressverschiebung spezifiziert ist.
LDAX B	LD A,(BC)	Der Inhalt des durch das Registerpaar BC adressierten Speicherplatzes wird in den Akkumulator (A-Register) geladen. Das C-Register beinhaltet die niederwertigen 8 Bit der Adresse und das Register B die hoeherwertigen 8 Bit.
LDAX D	LD A,(DE)	Der Inhalt des durch das Registerpaar DE adressierten Speicherplatzes wird in den Akkumulator (A-Register) geladen. Das Register E beinhaltet die niederwertigen 8 Bit der Adresse und das Register D die hoeherwertigen 8 Bit.
LDA nn	LD A,(nn)	Der Inhalt des durch nn adressierten Speicherplatzes wird in den Akkumulator (A-Register) geladen.
STA nn	LD (nn),A	Der Inhalt des Akkumulators (A-Register) wird auf den durch nn

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
		adressierten Speicherplatz transportiert.
STAX B	LD (BC),A	Der Inhalt des Akkumulators (A-Register) wird auf den Speicherplatz geladen, dessen Adresse im Registerpaar BC definiert ist. Das Register C beinhaltet die niederwertigen 8 Bit der Adresse und das Register B die hoeherwertigen 8 Bit.
STAX D	LD (DE),A	Der Inhalt des Akkumulators (A-Register) wird auf den Speicherplatz geladen, dessen Adresse im Registerpaar DE definiert ist. Das Register E beinhaltet die niederwertigen 8 Bit der Adresse und das Register D die hoeherwertigen 8 Bit.
---	LD A,I	Der Registerinhalt von I wird in den Akkumulator (A-Register) geladen. I ist das Interrupt-Register.
---	LD A,R	Der Registerinhalt von R wird in den Akkumulator (A-Register) geladen R ist das Refresh-Register.
---	LD I,A	Der Inhalt des Akkumulators (A-Register) wird in das Interrupt-Register geladen.
---	LD R,A	Der Akkumulatorinhalt wird in das Refresh-Register geladen.

9.6.1.2. 16-Bit-Ladebefehle

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
LXI dd,nn	LD dd,nn	Ein 16-Bit-Direktoperand wird in ein Doppelregister geladen.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
---	LD IX,nn	Ein 16-Bit-Direktoperand wird in das Indexregister IX geladen.
---	LD IY,nn	Ein 16-Bit-Direktoperand wird in das Indexregister IY geladen.
LHLD nn	LD HL,(nn)	Der Inhalt der durch nn und nn+1 adressierten Speicherplaetze wird in das Doppelregister HL geladen: Inhalt von nn+1 ---> Register H Inhalt von nn ---> Register L
---	LD dd,(nn)	Der Inhalt der durch nn und nn+1 adressierten Speicherplaetze wird in ein Doppelregister geladen: Inhalt von nn+1 ---> hoeherwertiges Register (B,D,SP_H) Inhalt von nn ---> niederwertiges Register (C,E,SP_L)
---	LD IX,(nn)	Der Inhalt der durch nn und nn+1 adressierten Speicherplaetze wird in das Indexregister IX geladen: Inhalt von nn+1 ---> Register IX_H Inhalt von nn ---> Register IX_L
---	LD IY,(nn)	Der Inhalt der durch nn und nn+1 adressierten Speicherplaetze wird in das Indexregister IY geladen: Inhalt von nn+1 ---> Register IY_H Inhalt von nn ---> Register IY_L
SHLD nn	LD (nn),HL	Der Inhalt des Doppelregisters HL wird auf die Adressen nn und nn+1 transportiert: Inhalt Reg. H --->Inhalt Adr.nn+1 Inhalt Reg. L --->Inhalt Adr.nn
---	LD (nn),dd	Der Inhalt eines Registerpaares wird auf die Adressen nn und nn+1 transportiert: Inhalt des hoeherwertigen Registers (B,D,SP_H) ---> Adr. nn+1

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
		Inhalt des niederwertigen Registers (C,E,SP_L) ---> Adr. nn
---	LD (nn),IX	Der Inhalt des Indexregisters IX wird auf die Adressen nn+1 und nn transportiert: Inhalt IX_H ---> Adr. nn+1 Inhalt IX_L ---> Adr. nn
---	LD (nn),IY	Der Inhalt des Indexregisters IY wird auf die Adressen nn+1 und nn transportiert: Inhalt IY_H ---> Adr. nn+1 Inhalt IY_L ---> Adr. nn
SPHL	LD SP,HL	Der Inhalt des Doppelregisters HL wird in den Stackpointer (Kellerzeiger) uebertragen: Inhalt Register H ---> SP_H Inhalt Register L ---> SP_L
---	LD SP,IX	Der Inhalt des Indexregisters IX wird in den Stackpointer (Kellerzeiger) uebertragen. Inhalt Register IX_H ---> SP_H Inhalt Register IX_L ---> SP_L
---	LD SP,IY	Der Inhalt des Indexregisters IY wird in den Stackpointer (Kellerzeiger) uebertragen: Inhalt Register IY_H ---> SP_H Inhalt Register IY_L ---> SP_L

Anmerkung:

Das gegebenenfalls auf den Operationscode unmittelbar folgende Byte ist das niederwertige Byte des 16-Bit-Wortes.

9.6.2. Indirekte Registeroperationen (PUSH- und POP-Befehle)

9.6.2.1. PUSH-Befehle

Bei den PUSH-Befehlen wird der Inhalt des Registerpaares qq oder des Registers IX bzw. IY in einen externen RAM-Keller (Stack) uebertragen, der als LIFO-Datei (letzte Eintragung wird zuerst gelesen) organisiert ist.
Der Stackpointer enthaelt dabei staendig eine aktuelle 16-Bit-Adresse, die der aktuell niedrigsten Adresse des Stackbereiches entspricht.

Der PUSH-Befehl subtrahiert 1 vom Inhalt des Stackpointers und laedt das hoeherwertige Byte des Registerpaares bzw. des Registers IX oder IY in die Speicherstelle, die durch den Inhalt des Stackpointers SP adressiert ist.
Danach wird der Inhalt des Stackpointers nochmals dekrementiert. Das niederwertige Byte wird jetzt in die Speicherstelle eingetragen, die durch den Inhalt des Stackpointers adressiert ist.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
PUSH PSW	PUSH AF	Erniedrigen SP (SP-1) <--- A Erniedrigen SP (SP-2) <--- F
PUSH B	PUSH BC	analog PUSH AF
PUSH D	PUSH DE	analog PUSH AF
PUSH H	PUSH HL	analog PUSH AF
---	PUSH IX	(SP-2) <--- IX_L (SP-1) <--- IX_H
---	PUSH IY	analog PUSH IX

9.6.2.2. POP-Befehle

Bei den POP-Befehlen wird der Inhalt der vom Stackpointer (Kellerzeiger) SP und (SP+1) adressierten 2 Bytes des externen Stacks in ein Registerpaar qq bzw. in ein Register IX oder IY uebertragen.

Der POP-Befehl uebertraegt zunaechst den Inhalt der Speicherstelle, die durch den aktuellen Wert des Stackpointers adressiert ist, in den niederwertigen Teil des Registerpaares bzw. des Registers IX oder IY.
Danach wird der Stackpointer inkrementiert, und der Inhalt der jetzt adressierten Speicherstelle wird in den hoeherwertigen Teil des Registerpaares bzw. Registers IX oder IY uebertragen. Der Stackpointer wird anschliessend erneut inkrementiert.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
POP PSW	POP AF	F <--- (SP) Erhoehen SP A <--- (SP+1) Erhoehen SP
POP B	POP BC	analog POP AF
POP D	POP DE	analog POP AF
POP H	POP HL	analog POP AF
---	POP IX	IX_H <--- (SP+1) IX_L <--- (SP)
---	POP IY	analog POP IX

Beispiel: Retten Register im Unterprogramm

```
BEISP: PUSH HL
       PUSH DE
       PUSH BC
         .          ) Unterprogramm-
         .          ) verarbeitungs-
         .          ) befehle

       PUSH BC
       PUSH DE
       PUSH HL
       RET
```

Nach Verarbeitung des Unterprogramms besitzen die Register (HL, DE, BC) wieder die gleichen Inhalte wie vor dem Aufruf.

9.6.3. Register-Austausch-Befehle

Die Register-Austausch-Befehle sind ein Byte lang, ausgenommen der zwischen (SP) und jeweils einem Basisregister. Sie besitzen eine Befehlslaenge von 2 Byte.
Durch die geringe Befehlslaenge werden kurze Interruptantwortzeiten moeglich. Fuer den Registeraustausch steht ein Hintergrundsatz der CPU-Register zur Verfuegung (auch 2. Registersatz genannt).

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
XCHG	EX DE,HL	Die 16-Bit-Inhalte der Registerpaare DE und HL werden ausgetauscht: DE <---> HL
---	EX AF,AF'	Die 16-Bit-Inhalte der Registerpaare AF und AF' werden ausgetauscht. AF' besteht aus den Registern A' und F' AF <---> AF'
---	EXX	Die 16-Bit-Inhalte der nachstehenden Register werden in folgender Weise getauscht: BC <---> BC' DE <---> DE' HL <---> HL'
XTHL	EX (SP),HL	Der Inhalt des Registers L wird gegen den Inhalt der Speicherstelle ausgetauscht, die durch den Inhalt des Stackpointers SP adressiert ist. Der Inhalt des Registers H wird gegen den Inhalt der Speicherstelle ausgetauscht, die durch den Inhalt des Stackpointers SP plus 1 adressiert ist. H <---> (SP+1) L <---> (SP)
---	EX (SP),IX	Der niederwertige Teil des Registers IX wird gegen den Inhalt der Speicherstelle ausgetauscht, die durch den Inhalt des Stackpointers SP adressiert ist. Der hoeherwertige Teil von IX wird gegen den Inhalt der Spei-

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
		cherstelle ausgetauscht, die durch den Inhalt SP plus 1 adressiert ist. IX_H <---> (SP+1) IX_L <---> (SP)
---	EX (SP),IY	analog EX (SP),IX IY_H <---> (SP+1) IY_L <---> (SP)

9.6.4. Blocktransportbefehle

Mit einem einzigen Befehl kann ein beliebig grosser Block des Speichers zu einem anderen Speicherplatz transportiert werden. Dieser Satz von Blocktransportbefehlen ist ausserordentlich wertvoll fuer die Verarbeitung grosser Datenbloecke.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle

---	LDIR	Transport mehrerer Datenbytes ab der Speicherstelle, die durch das Registerpaar HL adressiert wird nach der Speicherstelle, die durch das Registerpaar DE adressiert wird. Die Bytezahl ist im Registerpaar BC enthalten. Nach jeder Byteuebertragung wird der Inhalt von HL und DE um 1 erhoeht und BC um 1 vermindert. Die Uebertragung endet, wenn (BC)=Ø ist.
		Beispiel:
		LD HL,DATA ;Startadr. d. Quellber.
		LD DE,PUF ;Startadr. d. Zielber.
		LD BC,737 ;Laenge der Datenkette
		LDIR ;Transport der Daten-
		;kette nach Zielbereich
		;Erhoehen HL und DE,
		;Vermindern BC, Wieder-
		;holen bis BC=Ø.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle

---	LDI	Transport eines Datenbytes von der Speicherstelle, die durch das Register HL adressiert wird, nach der Speicherstelle, die durch das Register DE adressiert wird. Die Register DE und HL werden um 1 erhoeht, und das Register BC wird um 1 vermindert. Beispiel:
		LD HL,DATA;Startadr. d. Quellber.
		LD DE,PUF ;Startadr. d. Zielber.
		LD BC,132 ;Max. Kettenlaenge BC
		LD A,'K' ;Endemerkmal im Reg. A
		LOOP: CMP (HL);Vergleich Speicherinh.mit Endemerkm.
		JR Z,END-¤;Sprung zu END, wenn Zeichen gleich
		LDI ;Zeichentransport (HL) zu (DE)
		;Erhoehen HL und DE, Vermindern BC
		JP PE,LOOP;Sprung zu LOOP, wenn noch Zeichen zu transportieren sind, sonst gehe weiter.
		;P/V-Flag=1, solange BC≠Ø ist.
		END: JP HALT; Halt

---	LDDR	Der Befehl wirkt wie LDIR, nur werden die Register DE und HL um 1 vermindert.
---	LDD	Der Befehl wirkt wie LDI, nur werden die Register DE und HL um 1 vermindert.

9.6.5. Blocksuchbefehle

Diese Befehle sind fuer die Verarbeitung grosser Datenmengen geeignet. Mit einem einzigen Befehl kann ein Speicherblock beliebiger Groesse nach einem bestimmten 8-Bit-Zeichen durchsucht werden. Der Befehl ist automatisch beendet, wenn das Zeichen gefunden wurde.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
---	CPI	Vergleich des Inhalts des durch HL adressierten Speicherplatzes mit dem Inhalt des Akkumulators (A-Register). Das Register BC kann als Bytezaehler arbeiten. Das Register HL wird um 1 erhoeht. Das Registerpaar BC wird um 1 vermindert.
---	CPIR	Vergleich des Inhalts des Akkumulators mit dem Inhalt eines adressierten Speicherbereiches. Die Startadresse des Bereiches ist in dem Registerpaar HL enthalten, die Laenge des Bereiches in dem Registerpaar BC. Die zu suchende Konstante steht im Akkumulator. Der Vergleich endet, wenn der Akkumulator = (HL) ist oder wenn BC = Ø ist. Der Befehl sucht, indem er Register HL erhoeht und Register BC um 1 vermindert.
---	CPD	Der Befehl wirkt wie CPI, nur das Register HL wird vermindert.
---	CPDR	Der Befehl wirkt wie CPIR, nur das Register HL wird vermindert.

9.6.6. Arithmetische und logische Operationen

Die arithmetischen und logischen Befehle arbeiten mit Daten, die sich im Akkumulator (A-Register) und in anderen Universal-CPU-Registern oder auf den Speicherplaetzen befinden.
Die Ergebnisse dieser Operationen werden im Akkumulator untergebracht. Geeignete Flags werden entsprechend dem Ergebnis der Operationen gesetzt.

9.6.6.1. 8-Bit-Arithmetik

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
ADD r	ADD A,r	Der Registerinhalt r wird zum Akkumulatorinhalt addiert.
ADD M	ADD A,M	Der durch das Registerpaar HL adressierte Inhalt des Speicherplates M wird zum Inhalt des Akkumulators addiert.
ADI n	ADD A,n	Der Direktoperand n wird zum Inhalt des Akkumulators addiert.
---	ADD A,(IX+d)	Der Inhalt des durch Register IX plus Adresssverschiebung adressierten Speicherplatzes wird zum Inhalt des Akkumulators addiert. Das Ergebnis steht nach der Operation im Akkumulator.
---	ADD A,(IY+d)	Der Inhalt des durch Register IY plus Verschiebung adressierten Speicherplatzes wird zum Inhalt des Akkumulators addiert. Das Ergebnis steht nach der Operation im Akkumulator.
ADC r	ADC A,r	Der Registerinhalt r plus Carry-Flag (CY) wird zum Akkumulatorinhalt addiert.
ADC M	ADC A,M	Der durch das Registerpaar HL adressierte Inhalt des Speicherplatzes M plus Carry-Flag (CY) wird zum Inhalt des Akkumulators addiert.
ACI n	ADC A,n	Der Direktoperand n plus CY werden zum Inhalt des Akkumulators addiert.
---	ADC A,(IX+d)	Der Inhalt des durch Register IX plus Verschiebung d adressierten Speicherplatzes plus CY werden zum Akkumulator addiert.
---	ADC A,(IY+d)	Der Inhalt des durch Register IY plus Verschiebung d adressierten Speicherplatzes plus CY werden

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
		zum Akkumulators addiert.
SUB r	SUB r	Der Registerinhalt r wird vom Akkumulator subtrahiert.
SUB M	SUB M	Der durch Registerpaar HL adressierte Inhalt des Speicherplatzes M wird vom Inhalt des Akkumulators subtrahiert.
SUI n	SUB n	Der Direktoperand wird vom Inhalt des Akkumulators subtrahiert.
---	SUB (IX+d)	Der Inhalt des durch Register IX plus Verschiebung adressierten Speicherplatzes wird vom Inhalt des Akkumulators subtrahiert. (-128 ≤ d ≤ 127)
---	SUB (IY+d)	Der Inhalt des durch Register IY plus Verschiebung adressierten Speicherplatzes wird vom Inhalt des Akkumulators subtrahiert.
SBB r	SBC A,r	Das Carry-Flag wird zum Registerinhalt r addiert und dieses Ergebnis vom Inhalt des Akkumulators subtrahiert.
SBB M	SBC A,M	Das Carry-Flag wird zum Inhalt des durch Registerpaar HL adressierten Speicherplatzes M addiert und dieses Ergebnis wird vom Inhalt der Akkumulators subtrahiert
SBI n	SBC A,n	Der Direktoperand n plus CY wird vom Inhalt des Akkumulators subtrahiert.
---	SBC A,(IX+d)	Das Carry-Flag wird zum Inhalt des durch Register IX plus Verschiebung adressierten Speicherplatzes addiert und vom Inhalt des Akkumulators subtrahiert.
---	SBC A,(IY+d)	Das Carry-Flag wird zum Inhalt des durch Register IY plus Verschiebung adressierten Speicherplatzes addiert und vom Inhalt

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
		des Akkumulators subtrahiert.
INR r	INC r	Der Inhalt des Registers r wird um 1 erhoeht.
INR M	INC M	Der Inhalt des durch HL adressierten Speicherplatzes M wird um 1 erhoeht.
---	INC (IX+d)	Der Inhalt des durch IX plus Verschiebung adressierten Speicherplatzes wird um 1 erhoeht.
---	INC (IY+d)	Der Inhalt des durch IY plus Verschiebung adressierten Speicherplatzes wird um 1 erhoeht.
DCR r	DEC r	Der Inhalt des Registers r wird um 1 vermindert.
DCR M	DEC M	Der Inhalt des durch Registerpaar HL adresssierten Speicherplatzes wird um 1 vermindert.
---	DEC (IX+d)	Der Inhalt des durch IX plus Verschiebung adressierten Speicherplatzes wird um 1 vermindert.
---	DEC (IY+d)	Der Inhalt des durch IY plus Verschiebung adressierten Speicherplatzes wird um 1 vermindert.

9.6.6.2. 16-Bit-Arithmetik

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
DAD dd	ADD HL,dd	Der Inhalt von dd wird zum Inhalt des Registerpaares HL addiert.
---	ADD IX,IX	Der Inhalt des Registers IX wird mit sich selbst addiert. Diese Verdoppelung ist gleichbedeutend mit einer Linksverschiebung der

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
		16 Bit um eine Position.
---	ADD IY,IY	Der Inhalt des Registers IY wird mit sich selbst addiert. Diese Verdoppelung ist gleichbedeutend mit einer Linksverschiebung der 16 Bit um eine Bitposition.
---	ADD IX,pp	Der Inhalt von pp wird zum Inhalt des 16-Bit-Registers IX addiert.
---	ADD IY,pp	Der Inhalt von pp wird zum Inhalt des 16-Bit-Registers IY addiert.
---	ADC HL,dd	Der Inhalt von dd wird zum Inhalt des Registerpaares HL addiert. Der Inhalt des Carry-Flags wird ebenfalls addiert.
---	SBC HL,dd	Das Carry-Flag wird zu dem Doppelregister dd addiert und dieses Ergebnis vom Inhalt des Registerpaares HL subtrahiert.
INX dd	INC bb	Der Inhalt des Doppelregisters dd (bb) wird um 1 erhoeht.
DCX dd	DEC bb	Der Inhalt des Doppelregisters dd (bb) wird um 1 vermindert.

9.6.3. 8-Bit-Logikbefehle

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
ANA r ANA M ANI n	AND s	Logisches UND eines Registers, Speicherbytes oder Direktwertes mit dem Akkumulator. Das spezifizierte Byte wird bitweise mit dem Inhalt des Akkumulators konjunktiv verknuepft. Das logische UND zweier Bits ist nur dann 1, wenn beide Bits 1 sind.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
		Beispiel: Akkumulator: FCH 1111 11ØØ Register: ØFH ØØØØ 1111 ----------
		Resultat: ØCH ØØØØ 11ØØ
ORA r ORA M ORI n	OR s	Logisches ODER eines Registers, Speicherbytes oder Direktwertes mit dem Akkumulator. Das spezifizierte Byte wird bitweise mit dem Inhalt des Akkumulators disjunktiv verknuepft. Das logische ODER zweier Bits ist nur dann Ø, wenn beide Bits Ø sind. Beispiel: Akkumulator: FCH 1111 11ØØ Register: F1H 1111 ØØØ1 ----------
		Resultat: FDH 1111 11Ø1
XRA r XRA M XRI n	XOR s	Exklusives ODER eines Registers, Speicherbytes oder Direktwertes mit dem Akkumulator. Das spezifizierte Byte wird bitweise mit dem Inhalt des Akkumulators exklusiv verknuepft. Das exklusive ODER ist dann gleich 1, wenn ein Bit=1 und ein Bit=Ø ist. Beispiel: Akkumulator: FCH 1111 11ØØ Register: F1H 1111 ØØØ1 ----------
		Resultat: ØDH ØØØØ 11Ø1
CMP r CMP M CPI n	CP s	Der Inhalt von s (r, M, n) wird mit dem Akkumulator verglichen. Der urspruengliche Inhalt von A bleibt erhalten. Das Vergleichsergebnis ist durch Flags erkennbar.

9.6.7. Sprungbefehle

Es ist zwischen unbedingten und bedingten Spruengen zu unterscheiden. Es sind weiterhin relative Spruenge moeglich, die zur Adressenbildung anstelle von zwei Bytes nur eines benoetigen.

Bei bedingten Spruengen werden Sprungbedingungen getestet. Diese Bedingungen sind im Flagregister F enthalten.
In Abhaengigkeit von den Bedingungsflags koennen die Sprungbedingungen erfuellt sein oder nicht.
Bei einer erfuellten Sprungbedingung wird der Speicherzuordnungszaehler entsprechend der Adressenangabe im Sprungbefehl veraendert.
Bei nicht erfuellter Sprungbedingung wird der Sprungbefehl ignoriert.
Bei den relativen Spruengen wird das Sprungziel ueber den Wert e errechnet. Die Sprungweite e wird zum aktuellen Stand des Speicherzuordnungszaehlers addiert (im 2er-Komplement) und ermoeglicht einen Sprung im Bereich zwischen -128 und 127 Bytes.
Bei symbolischer Adressierung in relativen Spruengen berechnet der Assembler automatisch den Speicherzuordnungszaehler.

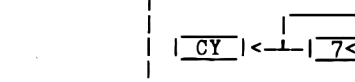
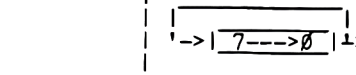
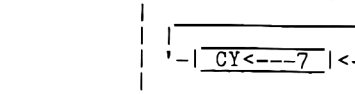
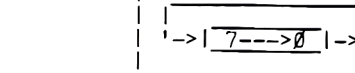
8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
JMP nn	JP nn	Unbedingter Sprung nach Adresse nn
JNZ nn	JP NZ,nn	Sprung nach Adresse nn, wenn Z-Flag gleich Ø ist.
JZ nn	JP Z,nn	Sprung nach Adresse nn, wenn Z-Flag gleich 1 ist.
JNC nn	JP NC,nn	Sprung nach Adresse nn, wenn C-Flag gleich Ø ist.
JC nn	JP C,nn	Sprung nach Adresse nn, wenn C-Flag gleich 1 ist.
JPO nn	JP PO,nn	Sprung nach Adresse nn, wenn P/V-Flag gleich Ø ist.
JPE nn	JP PE,nn	Sprung nach Adresse nn, wenn P/V-Flag gleich 1 ist.
JP nn	JP P,nn	Sprung nach Adresse nn, wenn S-Flag gleich Ø ist.
JM nn	JP M,nn	Sprung nach Adresse nn, wenn S-Flag 1 ist.
---	JR e	Unbedingter relativer Sprung.
——	JR NZ,e	Relativer Sprung um Verschiebung e, wenn Z-Flag gleich Ø ist.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
---	JR Z,e	Relativer Sprung um Verschiebung e, wenn Z-Flag gleich 1 ist.
---	JR NC,e	Relativer Sprung um Verschiebung e, wenn C-Flag gleich Ø ist.
---	JR C,e	Relativer Sprung um Verschiebung e, wenn C-Flag gleich 1 ist.
PCHL	JP M (HL)	Unbedingter Sprung zur Adresse, die im Register HL steht.
---	JP (IX)	Unbedingter Sprung zur Adresse, die im Register IX steht.
---	JP (IY)	Unbedingter Sprung zur Adresse, die im Register IY steht.
---	DJNZ e	Der Inhalt des Registers B wird um 1 vermindert. Bedingter relativer Sprungbefehl um Verschiebung e, wenn der Inhalt des Registers B ≠ Ø ist

9.6.8. Verschiebebefehle

Durch diese Befehle ist die Moeglichkeit gegeben, im Akkumulator (A-Register), in einem Universalregister oder in einem Speicherplatz Daten einfach oder zyklisch zu verschieben. Diese Operationen sind in einem externen grossen Gebiet einschliesslich der ganzzahligen Multiplikation und Division anwendbar.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
RLC	RLCA	Linksrotation des Akkumulatorinhalts. Der Inhalt des Akkumulators wird um eine Bitposition nach links verschoben. Das hoechstwertige Bit 7 wird zum Inhalt des niederwertigen Bits Ø

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
		und des Carry-Flags. 
RRC	RRCA	Rechtsrotation des Akkumulatorinhalts. Der Inhalt des Akkumulators wird um eine Bitposition nach rechts verschoben. Das niederwertige Bit Ø wird zum Inhalt des hoechstwertigen Bit 7 und des Carry-Flags. 
RAL	RLA	Linksrotation des Akkumulatorinhalts durch CY. Der Inhalt des Akkumulators wird um eine Bitposition nach links verschoben. Das hoechstwertige Bit 7 ersetzt das Carry-Flag, waehrend das Carry-Flag das niederwertigste Bit Ø des Akkumulators ersetzt. 
RAR	RRA	Rechtsrotation des Akkumulatorinhalts durch CY. Der Inhalt des Akkumulators wird um eine Bitposition nach rechts verschoben. Das niederwertigste Bit Ø ersetzt des Carry-Flag, waehrend das hoechstwertige Bit 7 durch das Carry-Flag ersetzt wird. 
---	RLC t	Linksrotation von t analog dem Befehl RLCA.
---	RRC t	Rechtsrotation von t analog dem Befehl RRCA.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
——	RL t	Linksrotation von t durch CY analog dem Befehl RLA.
——	RR t	Rechtsrotation von t durch CY analog dem Befehl RRA.
——	SLA t	Linksverschiebung von t um 1 Bit durch CY. Das niederwertige Registerbit Ø wird Ø. |CY|<—|7<---Ø|<—Ø
——	SRL t	Rechtsverschiebung von t um 1 Bit durch CY. Das hoechstwertige Registerbit 7 wird Ø. Ø—>|7--->Ø|-->|CY|
——	SRA t	Rechtsverschiebung von t um 1 Bit durch CY. Der Inhalt von Bit 7 bleibt erhalten. | '-|7--->Ø|-->|CY|
——	RLD	Zyklische Verschiebung nach links zwischen dem Akkumulator und dem Inhalt des durch HL adressierten Speicherplatzes. A (HL) |Ø1Ø1|1111| |ØØØØ|1111| vor dem Befehl |Ø1Ø1|ØØØØ| |1111|1111| nach dem Befehl Die unteren 4 Bit des durch HL adressierten Speicherplatzes werden in die oberen 4 Bitstellen uebertragen und diese ihrerseits in die unteren 4 Bitstellen des Akkumulators. Die unteren 4 Bits des Akkumulators werden in die unteren 4 Bits der Speicherstelle transportiert.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
---	RRD	Zyklische Verschiebung nach rechts zwischen dem Akkumulator und dem Inhalt des durch HL adressierten Speicherplatzes. A (HL) |Ø1Ø1|1111| |ØØØØ|1111| vor dem Befehl |Ø1Ø1|1111| |1111|ØØØØ| nach dem Befehl Die unteren 4 Bits der durch HL adressierten Speicherstelle werden in die unteren 4 Bitstellen des Akkumulators uebertragen und diese in die oberen der durch HL adressierten Speicherstelle. Die oberen 4 Bits aus der durch HL adressierten Speicherstelle werden in die unteren 4 Bitstellentransportiert.

9.6.9. Spezielle Akkumulator- und Flagbefehle

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
DAA	DAA	Korrigiert nach Addition / Subtraktion zweier gepackter BCD-Zahlen den Akkumulatorinhalt so, dass im Akkumulator wieder die gepackte BCD-Darstellung erreicht wird. Beispiel: 65 Ø11Ø Ø1Ø1 +57 Ø1Ø1 Ø111 --- 1Ø11 11ØØ ===

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
		Der DAA-Befehl fuehrt jetzt die notwendige Korrektur aus. In Abhaengigkeit von der Wertigkeit der zwei Halbbytes wird ein bestimmtes Korrekturbyte (im vorliegenden Beispiel 66) addiert. CY | ØØ11Ø Ø11Ø ↓ |---------- L | ØØ1Ø ØØ1Ø = 122
CMA	CPL	Bitweises Negieren (Komplementieren) des Akkumulatorinhaltes.
---	NEG	Subtrahieren des Akkumulator-Inhalts von Ø. Entspricht wertmaessig dem Zweierkomplement.
CMC	CCF	Komplementieren des Carry-Flags.
STC	SCF	Setzen des Carry-Flags.

9.6.10. Unterprogramm-Aufruf-Befehle

Diese Befehle sind eine Spezialform der Sprungbefehle. Es ist zwischen unbedingten und bedingten Unterprogrammaufrufen zu unterscheiden. Beim unbedingten Unterprogrammaufruf wird der dem Aufruf folgende Speicherplatzzuordnungszaehlerstand in den Stack gerettet.

Die im Befehl angegebene Unterprogrammstartadresse nn wird vom Speicherplatzzuordnungszaehler wieder mit der Absprungadresse aus dem Stack geladen.
Bei bedingten Unterprogrammaufrufen wird bei erfuellter Sprungbedingung analog dem unbedingten Unterprogrammaufruf verfahren. Bei nicht erfuellter Sprungbedingung wird der Befehl ignoriert.
Der hoeherwertige Adressteil im Speicherplatzzuordnungszaehler wird nach der Stackadresse minus 1 und der niederwertige Adressteil nach der Stackadresse minus 2 gebracht.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
CALL nn	CALL nn	Unbedingter Unterprogrammaufruf (SP-1) <--- PC_H (SP-2) <--- PC_L PC <--- nn
CNZ nn	CALL NZ,nn	Unterprogrammaufruf, wenn Z-Flag gleich Ø ist.
CZ nn	CALL Z,nn	Unterprogrammaufruf, wenn Z-Flag gleich 1 ist.
CNC nn	CALL NC,nn	Unterprogrammaufruf, wenn das C-Flag gleich Ø ist.
CC nn	CALL C,nn	Unterprogrammaufruf, wenn das C-Flag gleich 1 ist.
CPO nn	CALL PO,nn	Unterprogrammaufruf, wenn das P/V-Flag gleich Ø ist.
CPE nn	CALL PE,nn	Unterprogrammaufruf, wenn das P/V-Flag gleich 1 ist.
CP nn	CALL P,nn	Unterprogrammaufruf, wenn das S-Flag gleich Ø ist.
CM nn	CALL M,nn	Unterprogrammaufruf, wenn das S-Flag gleich 1 ist.
RST k	RST p	Der RST-Befehl ist ein spezieller Unterprogrammaufruf. Es sind folgenden 8 RST-Adressen zugelassen: p={ØØH; Ø8H; 1ØH; 18H; 28H; 3ØH; 38H}. Der hoeherwertige Adressteil ist dabei Ø. Der RST-Befehl entspricht in der weiteren Wirkung dem unbedingten Unterprogrammaufruf.

9.6.11. Unterprogramm-Ruecksprung-Befehle

Ein Ruecksprungbefehl beendet ein Unterprogramm. Es wird zwischen einem unbedingten Ruecksprung und bedingten Ruecksprungen aus Interrupt-Behandlungsroutinen unterschieden.

Bei einem unbedingten Rueecksprung und bei erfuellter Sprungbedingung bei bedingten Ruecksprueengen wird der beim Aufruf des Unterprogramms in den Stack gerettete Speicherzuordnungszaehlerinhalt wieder in den Speicherzuordnungszaehler zurueckgeschrieben.

PC_L <--- (SP)

PC_H <--- (SP+1)

SP <--- (SP+2)

Bei nichterfuellter Sprungbedingung wird der dem Ruecksprung folgende Befehl abgearbeitet.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
RET	RET	Unbedingter Ruecksprung
RNZ	RET NZ	Unterprogrammruecksprung, wenn das Z-Flag gleich Ø ist.
RZ	RET Z	Unterprogrammruecksprung, wenn das Z-Flag gleich 1 ist.
RNC	RET NC	Unterprogrammruecksprung, wenn das C=Flag gleich Ø ist.
RC	RET C	Unterprogrammruecksprung, wenn das C=Flag gleich 1 ist.
RPO	RET PO	Unterprogrammruecksprung, wenn das P/V-Flag gleich Ø ist.
RPE	RET PE	Unterprogrammruecksprung, wenn das P/V-Flag gleich 1 ist.
RP	RET P	Unterprogrammruecksprung, wenn das S-Flag gleich Ø ist.
RM	RET M	Unterprogrammruecksprung, wenn das S-Flag gleich 1 ist.
---	RETI	Es erfolgt ein Ruecksprung aus einer Interrupt-Behandlungsroutine.Dem Peripheriebaustein, der das Interrupt anmeldete, wird das Ende seines Programms mitgeteilt. Der Baustein gibt daraufhin die von ihm blockierte Interrupt-Kette (DAISY-CHAIN) wieder frei und

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
		ermoeglicht damit die Abarbeitung niederwertiger Interrupts. Der Inhalt von IFF2 wird nach IFF1 uebertragen. Durch die RETI-Anweisung wird der maskierbare Interrupt nicht freigegeben. Es sollte grundsaetzlich vor jeder RETI-Anweisung ein EI-Befehl stehen, der die Annahme spaeter folgender Interruptanforderungen ermoeglicht.
---	RETN	Es erfolgt ein Ruecksprung aus einer Interrupt-Behandlungsroutine, die durch einen nichtmaskierbaren Interrupt (NMI) ausgeloest wurde. Die Anweisung wirkt zunaechst wie die RET-Anweisung. Zusaetzlich wird der Inhalt von IFF2 nach IFF1 uebertragen, so dass die Abarbeitung maskierbarer Interrupt-Anforderungen unmittelbar nach Ausfuehrung des RETN-Befehls freigegeben ist, falls sie vor der NMI-Anforderung freigeben war.

9.6.12. CPU-Steuerbefehle

Diese Steuerbefehle loesen bei der CPU verschiedene Bedingungen und Betriebsarten aus.
Diese Gruppe enthaelt auch solche Befehle wie das Ein- und Ausschalten des Interrupt-Annahme-Flip-Flops oder das Setzen der Betriebsart Interruptverhalten.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
NOP	NOP	Die CPU fuehrt keine Operationen aus. Es werden Refresh-Zyklen erzeugt.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
HLT	HALT	Die CPU fuehrt solange eine Folge von NOP-Befehlen aus, bis ein Interrupt oder der RESET-Eingang an der CPU aktiv wird. Es werden Refresh-Zyklen erzeugt.
DI	DI	Der maskierbare Interrupt wird durch Ruecksetzen der Interrupt-Freigabe-Flip-Flops IFF1 bzw. IFF2 der CPU gesperrt. Nichtmaskierbare Interrupts werden anerkannt.
EI	EI	Der maskierbare Interrupt wird durch Setzen der Interrupt-Freigabe-Flip-Flops IFF1 bzw. IFF2 der CPU freigegeben. Waehrend der Ausfuehrung des Befehls akzeptiert die CPU keine Interruptanforderungen.
---	IM Ø	Der Befehl bringt die CPU in den Interruptmodus Ø.
---	IM 1	Der Befehl bringt die CPU in den Interruptmodus 1.
---	IM 2	Der Befehl bringt die CPU in den Interruptmodus 2.

9.6.13. Bittest- und -Setzbefehle (Bitmanipulation)

Die Bitmanipulationsbefehle erlauben, Bits in einem Register oder auf einem Speicherplatz zu setzen, zu loeschen und zu testen.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
---	SET b,r	Die durch b gekennzeichnete Bitposition wird in dem Register r gesetzt.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
---	SET b,M	Die durch b gekennzeichnete Bitposition wird in der Speicherstelle gesetzt, die durch das Register HL adressiert ist.
---	SET b,(IX+d)	Die durch b gekennzeichnete Bitposition wird in der Speicherstelle gesetzt, die durch den Inhalt des Indexregisters IX plus Verschiebung adressiert ist.
---	SET b,(IY+d)	Die durch b gekennzeichnete Bitposition wird in der Speicherstelle gesetzt, die durch den Inhalt des Indexregisters IY plus Verschiebung adressiert ist.
---	RES b,t	Die durch b gekennzeichnete Bitposition in t wird geloescht.
---	BIT b,t	Die durch b gekennzeichnete Bitposition in t wird getestet. Das Komplement des zu testenden Bits wird in das Z-Flag geladen.

9.6.14. Eingabebefehle

Die Ein- und Ausgabegruppe gestattet einen weiteren Anwendungsbereich von Datentransfer zwischen Speicherplaetzen oder den Universalregistern der CPU und den externen E/A-Geraeten. Die Eingabebefehle setzen automatisch das Flagregister, so dass keine zusaetzlichen Befehle noetig sind, um den Status der Eingabedaten zu ermitteln.

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
IN n	IN A,(n)	Kanaladresse wird mittels Direktoperand eingestellt. Zielregister ist der Akkumulator A <--- (n)

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
---	IN r,(C)	Kanaladresse wird indirekt ueber das Register C eingestellt. r <--- (C)
---	INI	Kanaladresse wird indirekt ueber Register C eingestellt. Zieladr. ueber Register HL. B kann als Bytezaehler arbeiten. B wird dekrementiert, HL inkrementiert. (HL) <--- (C) B <--- B-1 HL <--- HL+1
---	INIR	Kanaladresse wird indirekt ueber Register C eingestellt, Zieladr. ueber Register HL. B arbeitet als Bytezaehler. B wird dekrementiert, HL inkrementiert. Es wird eine Blockuebertragung durchgefuehrt bis B=Ø ist. (HL) <--- (C) B <--- B-1 HL <--- HL+1 Wiederholen bis B=Ø.
---	IND	Kanaladresse wird indirekt ueber Register C eingestellt. Zieladr. ueber Register HL. B kann als Bytezaehler arbeiten. B und HL werden dekrementiert. (HL) <--- (C) B <--- B-1 HL <--- HL-1
---	INDR	Kanaladresse wird indirekt ueber Register C eingestellt. Zieladr. ueber Register HL. B arbeitet als Bytezaehler. B und HL werden dekrementiert. Es wird eine Blockuebertragung durchgefuehrt, bis B = Ø ist. (HL) <--- (C) B <--- B-1 HL <--- HL-1 Wiederholen B = Ø.

Die Kanaladresse liegt auf der unteren Haelfte des Adressenbusses AØ - A7. Auf der oberen Haelfte des Adressenbusses liegt bei IN A,(n) der Akkumulatorinhalt bei den restlichen Befehlen der Inhalt des Registers B.

9.6.15. Ausgabebefehle

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
OUT n	OUT (n),A	Kanaladresse wird mit Direktoperand eingestellt. Quellregister ist der Akkumulator: (n) <--- A.
---	OUT C,r	Kanaladresse wird indirekt ueber Register C eingestellt. Quellregister ist r: (C) <--- r.
---	OUTI	Kanaladresse wird indirekt ueber das Register C eingestellt, Quelladresse ueber Register HL. B kann als Bytezaehler arbeiten. B wird dekrementiert, HL inkrementiert. (C) <--- (HL) B <--- B-1 HL <--- HL+1
---	OTIR	Kanaladresse wird indirekt ueber das Register C eingestellt, Quelladresse ueber Register HL. B arbeitet als Bytezaehler. B wird dekrementiert, HL inkrementiert. Es wird eine Blockuebertragung durchgefuehrt, bis B = Ø ist. (C) <--- (HL) B <--- B-1 HL <--- HL+1 Wiederholen bis B = Ø
---	OUTD	Kanaladresse wird indirekt ueber Register C eingestellt, Quelladresse ueber Register HL. B kann als Bytezaehler arbeiten. B und HL werden dekrementiert. (C) <--- (HL) B <--- B-1 HL <--- HL-1

8080-Mnemonik	Z80-Mnemonik	Wirkungsweise der Befehle
---	OTDR	Kanaladresse wird indirekt ueber das Register C eingestellt, Quelladresse ueber Register HL. B arbeitet als Bytezaehler. B und HL werden dekrementiert. Es wird eine Blockuebertragung durchgefuehrt, bis B=Ø ist. (C) <--- (HL) B <--- B-1 HL <--- HL-1 Wiederholen bis B=Ø ist.

9.6.16. Abkuerzungsverzeichnis zur Befehlsbeschreibung

r,r' : Register A, B, C, D, E, H und L koennen eingesetzt werden.

dd : Doppelregister BC, DE, HL und SP koennen eingesetzt werden.

qq : Doppelregister AF, BC, DE und HL sind moeglich.

pp : Doppelregister BC, DE und SP sind erlaubt.

bb : Doppelregister BC, DE, HL, SP, IX und IY sind erlaubt.

s : r, n, M, (IX+d) und (IY+d) sind erlaubt.

t : r, M, (IX+d) und (IY+d) sind moeglich.

n : 8-Bit-Direktoperand

nn : 16-Bit-Direktoperand

d : Verschiebung bei Adressierung ueber Indexregister, erlaubt im Bereich von $-128 \leq d \leq 127$
Die Bereichsgrenzen werden vom Assembler nicht geprueft!

e : relative Sprungsadresse,
erlaubt im Bereich von $-128 \leq e \leq 127$
Die Bereichsgrenzen werden vom Assembler nicht geprueft!

b : Bit, das in den Bitmanipulierbefehlen behandelt werden soll $Ø \leq b \leq 7$

M : Inhalt des durch HL adressierten Speicherplatzes

k : Die Werte Ø, 1, 2, 3, 4, 5, 6, 7 sind erlaubt.

p : Die Werte ØØH, Ø8H, 1ØH, 18H, 28H, 3ØH, 38H sind erlaubt.

CY : Carry-Flag

Anmerkung:

In der 8080-Mnemonik wird fuer die Operanden nur der erste Buchstabe geschrieben:

. H fuer HL
. D fuer DE
. B fuer BC.

Aber es bleibt SP, und fuer AF wird PSW verwendet.
Fuer M darf nicht (HL) geschrieben werden.

6.6.17. Arbeit mit den Bedingungsbits (Flags)

Das Flagregister F gibt Auskunft ueber das Ergebnis der letzten Prozessoroperation.
Es dient im wesentlichen dazu, bedingte Programmverzweigungen bzw. bedingte Unterprogrammaufrufe oder -ruecksprueng auszufuehren.

Flagregister

7	6	5	4	3	2	1	Ø
S	Z	X	H	X	P/V	N	CY

S - Vorzeichenbit (Sign - Flag)
Z - Nullbit (Zero - Flag)
H - Halbbyteueberlaufbit (Half - Carry - Flag)
N - Additions- / Subtraktionsbit
P/V - Paritaets- / Uebertragsbit (Parity / Overflow - Flag)
CY - Uebertragsbit (Carry - Flag)
X - nicht belegt

Bemerkung:

Im Testhilfsprogramm DU wird fuer das H-Flag I, fuer das P/V-Flag E und fuer das CY-Flag geschrieben.

Vorzeichenbit (S-Flag)

Bei bestimmten Befehlen wird das hoechstwertige Bit des Akkumulators geladen.
Bei Ausfuehrung von arithmetischen Befehlen mit vorzeichenbehafteten Zahlen wird eine positive Zahl durch eine Ø und eine negative Zahl durch eine 1 in der hoechstwertigen Bitstelle gekennzeichnet.

Nullbit (Z-Flag)

Es wird bei arithmetischen und logischen 1-Byte-Operationen gesetzt, wenn das Ergebnisbyte des Akkumulators Ø ist. Sonst wird das Ergebnisbyte zurueckgesetzt.

Bei Vergleichs- und Suchbefehlen wird das Z-Flag gesetzt, sobald der Vergleich positiv ausgeht.

Bei den Bit-Befehlen wird das Z-Flag **mit dem komplementaeren Wert** des getesteten Bits geladen.

Bei der Uebertragung eines Bytes zwischen einer Speicherstelle und einer E/A-Schnittstelle (INI, IND, OUTI, OUTD) wird das Z-Flag 1 gesetzt, wenn der Wert des Zaehlregisters Ø wird.
Bei IN r,(C) wird das Z-Flag gesetzt, wenn die eingezogenen bzw. am E/A-Tor anliegenden Daten den Wert Ø haben.

Halbbyte-Uebertragsbit (H-Flag)

Das H-Flag wird entsprechend dem Uebertragungsergebnis zwischen den Bits 3 und 4 einer arithmetischen 1-Byte-Operation gesetzt (falls Uebertrag) oder rueckgesetzt (falls kein Uebertrag). Es wird beim Befehl DAA verwendet, um das Ergebnis einer gepackten BCD-Addition bzw. -Subtraktion zu korrigieren.

H	Addition	Subtraktion
1	Uebertrag von Bit 3 zu Bit 4	Negativer Uebertrag von Bit 4
Ø	Kein Uebertrag von Bit 3 zu Bit 4	Kein negativer Uebertrag von Bit 4

Paritaets- / Ueberlaufbit (P/V-Flag)

Es wird unterschiedlich genutzt.
Bei arithmetischen Befehlen wird das P/V-Flag gesetzt, wenn im Ergebnis das hoechste Bit des Akkumulators gesetzt wird.

Bei logischen Operationen und Verschiebebefehlen dient das P/V-Flag zur Ueberpruefung der Paritaet des Ergebnisses. Ist die Anzahl der gesetzten Bits im angesprochenen Byte gerade (Ø, 2, 4, 6, 8), so wird das P/V-Flag gesetzt, ansonsten rueckgesetzt.

Bei Blocktransport- (LDI, LDIR, LDD, LDDR) und Blocksuchbefehlen (CPI, CPIR, CPD, CPDR) gibt das P/V-Flag Auskunft ueber den Stand des Bytezaehlers. Das P/V-Flag wird rueckgesetzt, wenn nach Dekrementieren des Bytezaehlers (=<BC>) als Ergebnis Ø entsteht. In allen uebrigen Faellen bleibt das P/V-Flag 1.

Additions- / Subtraktionsbit (N-Flag)

Es wird intern bei dem DAA-Befehl benutzt, um zwischen Additions- und Subtraktionsbefehlen zu unterscheiden. Bei allen Additionsbefehlen wird das N-Flag rueckgesetzt. Subtraktionsbefehle setzten das N-Flag.

Uebertragsbit (CY-Flag)

Das Setzen / Ruecksetzen des CY-Flags wird je nach ausgefuehrter Operation verschieden behandelt.

Das CY-Flag wird gesetzt, wenn

- bei Additionsbefehlen ein Uebertrag und
- bei Subtraktionsbefehlen ein negativer Uebertrag entsteht.

Es wird rueckgesetzt,

- bei Additionsbefehlen, die keinen Uebertrag erzeugen und
- bei Subtraktionsbefehlen, die keinen negativen Uebertrag erzeugen.

Bei Verschiebebefehlen (RLA, RLC, RL, RR) wird das CY-Flag als Zwischenspeicher fuer die Uebertragung des niederwertigsten bzw. hoechstwertigen Bits eines CPU-Registers bzw. Speicherplatzes benutzt.

Bei den Befehlen RLCA, RLC und SLA enthaelt das CY-Flag den Wert des hoechstwertigen Bits, das durch den Befehl aus dem behandelten Register bzw. Speicherplatz hinausgeschoben wurde.

Bei den Befehlen RRCA, RRC, SRA, SRL enthaelt das CY-Flag analog den Wert des niederwertigsten Bits.

Die logischen Befehle AND, OR, XOR setzen das CY-Flag grundsaetzlich zurueck.

Die speziell fuer das CY-Flag vorgesehenen Befehle SCF (Setzen CY-Flag) und CCF (Komplement C-Flag) erlauben das Setzen bzw. Komplementieren des CY-Flags.

9.7. Cross-Referenz

Das Cross-Referenz-Programm verarbeitet eine speziell uebersetzte Druckdatei zu einer Liste aller Adressverweise innerhalb des Moduls einschliesslich der Speicherplaetze, wo sie definiert sind.
Die Cross-Referenz kann als Hilfsmittel beim Programmtest genutzt werden.
Das Cross-Referenz-Programm erlaubt dem Programmierer das Verarbeiten einer vom Assembler erzeugten Cross-Referenz-Datei.
Diese Datei enthaelt Steuerzeichen, die waehrend der Uebersetzung mit dem Assembler eingesetzt wurden.

Das Cross-Referenz-Programm erzeugt eine Liste, die der .PRN-Datei gleicht und zwei zusaetzliche Merkmale besitzt:

1. Jede Quellanweisung wird mit einer Cross-Referenz-Nummer versehen.

2. Am Listenende erscheinen die Variablennamen in alphabetischer Reihenfolge.
   Jedem Namen werden die Nummer der Zeilen in aufsteigender Folge zugeordnet, wo er auftritt. Die Nummer der Zeile, wo er definiert wurde, ist durch # gekennzeichnet.

Die Cross-Referenz-Liste ersetzt die .PRN-Datei des Assemblers und erhaelt auch den Dateityp .PRN.

9.7.1. Erzeugen einer Cross-Referenz-Liste

Das Erzeugen einer Cross-Referenz-Liste erfolgt in zwei Schritten:

1. Erzeugen einer Cross-Referenz-Datei (.CRF)
2. Generieren einer Cross-Referenz-Liste (.PRN)

Der erste Schritt wird im Assembler ausgefuehrt; der zweite im Cross-Referenz-Programm.

9.7.1.1. Erzeugen einer Cross-Referenz-Datei

Fuer das Erzeugen einer Cross-Referenz-Datei muss in der Kommandozeile des Assemblers der /C-Schalter gesetzt werden.

Beispiel:

```
ASM =PROG/C
```

damit wird PROG.MAC uebersetzt und PROG.REL (Objektdatei) und PROG.CRF (Cross-Referenz-Datei) erzeugt.

9.7.1.2. Generieren der Cross-Referenz-Liste

Die Cross-Referenz-Liste wird beim Durchlaufen der .CRF-Datei durch das Cross-Referenz-Programm erzeugt.

Der Aufruf erfolgt mit

```
REF
```

Das Programm meldet sich mit einem Stern (*).
Nun ist der Name der .CRF-Datei einzugeben:

```
=dateiname
```

Beispiel:

```
REF =PROG
```

es wird die Datei PROG.PRN erzeugt, d.h. die gesamte Assemblerliste bestehend aus uebersetztem Code, Systemtabelle und Crossreferenz.
Diese .PRN-Datei kann unter Nutzung der Betriebssystemkommandos gedruckt oder auf dem Bildschirm angezeigt werden.
Zusaetzlich unterstuetzt REF die gleichen Ausgabegeraete wie der Assembler.

Beispiel:

```
REF LST:=PROG
```

gibt die Liste auf den Drucker aus. Es wird keine Diskettendatei erzeugt.

```
REF TTY:=PROG
```

Die Ausgabe erfolgt nur auf dem Bilschirm.

Es ist auch moeglich, ein anderes Laufwerk fuer die Ausgabe anzugeben.

Beispiel:

```
REF B:=A:PROG
```

Die Diskette mit PROG.CRF befindet sich im Laufwerk A; PROG.PRN soll auf die Diskette gebracht werden, die im Laufwerk B liegt.

Nach Beenden meldet sich REF mit einem Stern (*) zurueck, es kann ein neuer Dateiname eingegeben werden.

Die Rueckkehr zum Betriebssytem erfolgt mittels ^C.

Die Druckdatei kann auch einen anderen Namen und einen anderen Dateityp erhalten, dann sind diese anzugeben.

Beispiel:

```
     REF   PROG.CRL=PROG
bzw.
     REF   PROGREF=PROG
```

Die obere Kommandozeile erzeugt eine Druckdatei mit dem Namen PROG.CRL; die untere generiert eine Datei mit der Bezeichnung PROGREF.PRN.
Dateitypen koennen angegeben werden, um die Cross-Referenz-Liste von der Druckdatei des Assemblers zu unterscheiden.

9.7.2. Pseudooperationen zur Listensteuerung

Die Option fuer das Erzeugen einer Cross-Referenz kann fuer Programmabschnitte aber nicht fuer das ganze Programm angewendet werden.
Zum Steuern des Auflistens oder Weglassens der Cross-Referenz sind in der Quelldatei die Pseudooperationen .CREF und .XCREF anzugeben. Sie koennen an jeder beliebigen Stelle im Programm in das Operationsfeld geschrieben werden.

.CREF und .XCREF haben keine Argumente.

Pseudooperation	Bedeutung
.CREF	Erzeugen der Cross-Referenz .CREF ist die Standardbedingung: .CREF ist zu verwenden, wenn nach einem .XCREF das Erzeugen der Cross-Referenz wieder gestattet werden soll. .CREF wirkt, bis ein .XCREF erscheint. Bemerkung: .CREF wirkt nur, wenn in der Kommandozeile des Assemblers der /C-Schalter gesetzt wurde.
.XCREF	Unterdruecken der Cross-Referenz .XCREF schaltet die Wirkung von .CREF aus. .XCREF unterdrueckt die Cross-Referenz fuer Programmabschnitte. Weil weder .CREF noch .XCREF Wirkung haben, wenn nicht der /C-Schalter in der Assembler-Kommandozeile gesetzt ist muss .XCREF nicht verwendet werden, wenn eine normale Liste (ohne Cross-Referenz) gewuenscht wird. Das wird auch erreicht einfach durch Weglassen des /C-Schalters in der Kommandozeile.

9.8. Uebersicht ueber die Pseudooperationen des Assemblers

9.8.1. Einzelfunktions-Pseudooperationen

Pseudooperationen zur Auswahl der Anweisungsliste

	Seite
.Z80	23
.8080	23

Pseudooperationen zur Daten- und Symboldefinition

	Seite
<name> ASET <exp>	28
BYTE EXT <symbol>	26
BYTE EXTRN <symbol>	26
BYTE EXTERNAL <symbol>	26
DB <exp>[,<exp>...]	24
DB <string>[,<string>...]	24
DC <string>	24
DEFB <exp>[,<exp>...]	24
<name> DEFL <exp>	28
DEFM <string>[,<string>...]	24
DEFS <exp>[,<exp>...]	25
DEFW <exp>[,<exp>...]	26
DS <exp>[,<val>]	25
DW <exp>[,<exp>...]	26
ENTRY <name>[,<name>...]	27
<name> EQU <exp>	26
EXT <name>,[<name>...]	26
EXTRN <name>[,<name>...]	26
EXTERNAL <name>[,<name>...]	26
GLOBAL <name>[,<name>...]	27
PUBLIC <name>[,<name>...]	27
<name> SET <exp> (Nicht fuer Z80-Anweisungsliste!)	28

Pseudooperationen zur Zuweisung des Speicherzuordnungszaehlers

	Seite
ASEG	29
CSEG	29
DSEG	30
COMMON /<blockname>/	31
ORG	32
.PHASE <exp>	32
.DEPHASE	32

Dateibezogene Pseudooperationen

	Seite
.COMMENT <delim><text><delim>	33
END [<exp>]	34
INCLUDE <filename>	35
¤INCLUDE <filename>	35
MACLIB <filename>	35
NAME ("modulname")	35
.RADIX <exp>	36
.REQUEST <filename>[,<filename>...]	36

9.8.2. Pseudooperationen zur Listensteuerung

Pseudooperationen zur Formatsteuerung

	Seite
*EJECT [<exp>] (ein * ist Teil der Pseudooperation)	38
¤EJECT	38
PAGE <exp>	38
SUBTTL [<text>]	38
TITLE <text>	38
¤TITLE [<text>]	38
	38

Allgemeine Pseudooperationen zur Listensteuerung

	Seite
.LIST	39
.XLIST	39
.PRINTX <delim><text><delim>	40

Pseudooperationen zur bedingten Listensteuerung

	Seite
.SFCOND	40
.LFCOND	40
.TFCOND	41

Pseudooperationen zur Listensteuerung der Makroerweiterung

	Seite
.LALL	41
.SALL	41
.XALL	41

Pseudooperationen zur Steuerung der Cross-Referenz

	Seite
.XCREF	42
.CREF	42

9.8.3. Pseudooperationen zur Makrofaehigkeit

Pseudooperationen zur Makrodefinition

		Seite
	<name> MACRO <parameter>[,<parameter>...]	43
	ENDM	48
	EXITM	48
	LOCAL <parameter>[,<parameter>...]	49

Wiederholungspseudooperationen

		Seite
	REPT <exp>	46
	IRP <dummy>,<parameter in spitze Klammern eingeschlossen>	46
	IRPC <dummy>,<string>	47

Pseudooperationen zur bedingten Assemblierung

		Seite
*	COND <exp>	52
	ELSE	54
*	ENDC	54
	ENDIF	54
	IF <exp>	52
	IF <arg>	52
	IFB <arg>	53
	IFDEF <symbol>	53
	IFDIF <arg1>,<arg2>	53
	IFE <exp>	53
	IFF <exp>	53
	IFIDN <arg1>,<arg2>	53
	IFNB <arg>	53
	IFNDEF <symbol>	53
	IFT <exp>	52
	IF1	53
	IF2	53

9.9 Mikrobefehlsliste - Z80 - Operationscodes

Mikrobefehlsliste - Z8Ø Operationscodes — Blatt 1

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
8-Bit-Ladebefehle						
LD r,r'	r <-- r'		r/r' | A B C D E H L A | 7F 78 79 7A 7B 7C 7D B | 47 4Ø 41 42 43 44 45 C | 4F 48 49 4A 4B 4C 4D D | 57 5Ø 51 52 53 54 55 E | 5F 58 59 5A 5B 5C 5D H | 67 6Ø 61 62 63 64 65 L | 6F 68 69 6A 6B 6C 6D			.....
LD r,n	r <-- n		r | A B C D E H L 3E Ø6 ØE 16 1E 26 2E	n		.....
LD r,(HL)	r <-- (HL)		r | A B C D E H L 7E 46 4E 56 5E 66 6E			.....
LD r,(IX+d)	r <-- (IX+d)	DD	r | A B C D E H L 7E 46 4E 56 5E 66 6E	d		.....

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
LD r,(IY+d)	r <-- (IY+d)		r: A B C D E H L			
		FD	7E 46 4E 56 5E 66 6E	d		.....
LD (HL),r	(HL) <-- r		r: A B C D E H L			
			77 7Ø 71 72 73 74 75			.....
LD (IX+d),r	(IX+d)<-- r		r: A B C D E H L			
		DD	77 7Ø 71 72 73 74 75	d		.....
LD (IY+d),r	(IY+d)<-- r		r: A B C D E H L			
		FD	77 7Ø 71 72 73 74 75	d		.....
LD (HL),n	(HL) <-- n		36	n		.....
LD (IX+d),n	(IX+d)<-- n	DD	36	d	n	.....
LD (IY+d),n	(IY+d)<-- n	FD	36	d	n	.....
LD A,(dd)	A <-- (dd)		(dd): (BC) (DE)			
			ØA 1A			.....

Mikrobefehlsliste – Z8Ø Operationscodes — Blatt 3

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
LD (dd),A	(dd) <-- A		(dd): (BC) (DE)			
			Ø2 12			.....
LD A,($n_L n_H$)	A <-- ($n_L n_H$)		3A	n_L	n_H	
LD ($n_L n_H$),A	($n_L n_H$) <-- A		32	n_L	n_H	.....
LD A,I	A <-- I	ED	57			.|||ØØ
LD I,A	I <-- A	ED	47			.....
16-Bit-Ladebefehle						
LD dd,$n_L n_H$	dd <-- $n_L n_H$		dd: BC DE HL SP			
			Ø1 11 21 31	n_L	n_H	.....
LD IX,$n_L n_H$	IX <-- $n_L n_H$	DD	21	n_L	n_H	.....
LD IY,$n_L n_H$	IY <-- $n_L n_H$	FD	21	n_L	n_H	.....
LD dd,($n_L n_H$)	dd_H <-- (nn+1)		dd: BC DE HL SP			
	dd_L <-- (nn)	ED	4B 5B 6B 7B	n_L	n_H	.....

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNC V
LD HL,($n_L n_H$)	H <-- (nn+1) L <-- (nn)		2A	n_L	n_H	.....
LD IX,($n_L n_H$)	IX_H <-- (nn+1) IX_L <-- (nn)	DD	2A	n_L	n_H	.....
LD IY,($n_L n_H$)	IY_H <-- (nn+1) IY_L <-- (nn)	FD	2A	n_L	n_H	.....
LD ($n_L n_H$),dd	(nn+1)<-- dd_H (nn) <-- dd_L	ED	dd | BC DE HL SP ---- | ---------- 43 53 63 73	n_L	n_H	.....
LD ($n_L n_H$),HL	(nn+1)<-- H (nn) <-- L		22	n_L	n_H	.....
LD ($n_L n_H$),IX	(nn+1)<-- IX_H (nn) <-- IX_L	DD	22	n_L	n_H	.....

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNC V
LD ($n_L n_H$),IY	(nn+1)<-- IY_H	FD	22	n_L	n_H	.....
	(nn) <-- IY_L					
LD SP,HL	SP <-- HL		F9			.....
LD SP,IX	SP <-- IX	DD	F9			.....
LD SP,IY	SP <-- IY	FD	F9			.....
PUSH dd	(SP-2)<-- dd_L		dd | BC DE HL AF			
	(SP-1)<-- dd_H		C5 D5 E5 F5			.....
PUSH IX	(SP-2)<-- IX_L	DD	E5			.....
	(SP-1)<-- IX_H					
PUSH IY	(SP-2)<-- IY_L	FD	E5			.....
	(SP-1)<-- IY_H					

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNC V
POP dd	dd_H <-- (SP+1)		dd | BC DE HL AF			
	dd_L <-- (SP)		C1 D1 E1 F1			.....
POP IX	IX_H <-- (SP+1)	DD	E1			.....
	IX_L <-- (SP)					
POP IY	IY_H <-- (SP+1)	FD	E1			.....
	IY_L <-- (SP)					
Austauschbefehle						
EX DE,HL	DE <--> HL		EB			.....
EX (SP),HL	H <--> (SP+1) L <--> (SP)		E3			.....
EX (SP),IX	IX_H <--> (SP+1)	DD	E3			.....
	IX_L <--> (SP)					

Mikrobefehlsliste – Z8Ø Operationscodes Blatt 7

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
EX (SP),IY	IY_H <--> (SP+1) IY_L <--> (SP)	FD	E3			.....
EX AF,AF'	AF <--> AF'		Ø8			.....
EXX	BC <--> BC' DE <--> DE' HL <--> HL'		D9			.....
Blocktransportbefehle						
LDI	(DE) <-- (HL) DE <-- DE+1 HL <-- HL+1 BC <-- BC-1	ED	AØ			..Ø|Ø. a
LDIR	s.LDI bis BC = Ø	ED	BØ			..ØØØ.
LDD	(DE) <-- (HL) DE <-- DE-1 HL <-- HL-1 BC <-- BC-1	ED	A8			..Ø|Ø. a

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
LDDR	s.LDD bis BC = Ø	ED	B8			..ØØØ.
Suchbefehle						
CPI	A (HL) HL <-- HL+1 BC <-- BC-1	ED	A1			‖‖‖1. b a
CPIR	s.CPI bis BC = Ø oder A = (HL)	ED	B1			‖‖‖1. b a
CPD	A (HL) HL <-- HL-1 BC <-- BC-1	ED	A9			‖‖‖1. b a
CPDR	s.CPD bis BC = Ø oder A = (HL)	ED	B9			‖‖‖1. b a
8-Bit-Arithmetik-Befehle						
ADD A,r	A <-- A+r		r | A B C D E H L | 87 8Ø 81 82 83 84 85			‖‖VØ‖

Befehl	Operation		Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
ADC A,r	A	<-- A+r+CY		r | A B C D E H L | 8F 88 89 8A 8B 8C 8D			|||VØ|
ADD A,n	A	<-- A+n		C6	n		|||VØ|
ADC A,n	A	<-- A+n+CY		CE	n		|||VØ|
ADD A,(HL)	A	<-- A+(HL)		86			|||VØ|
ADD A,(IX+d)	A	<-- A+(IX+d)	DD	86	d		|||VØ|
ADD A,(IY+d)	A	<-- A+(IY+d)	FD	86	d		|||VØ|
ADC A,(HL)	A	<-- A+(HL)+CY		8E			|||VØ|
ADC A,(IX+d)	A	<-- A+(IX+d)+CY	DD	8E	d		|||VØ|
ADC A,(IY+d)	A	<-- A+(IY+d)+CY	FD	8E	d		|||VØ|
SUB r	A	<-- A-r		r | A B C D E H L | 97 9Ø 91 92 93 94 95			|||V1|

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNC V
SBC A,r	A <-- A-r-CY		r | A B C D E H L | 9F 98 99 9A 9B 9C 9D			|||V1|
SUB n	A <-- A-n		D6	n		|||V1|
SBC A,n	A <-- A-n-CY		DE	n		|||V1|
SUB (HL)	A <-- A-(HL)		96			|||V1|
SUB (IX+d)	A <-- A-(IX+d)	DD	96	d		|||V1|
SUB (IY+d)	A <-- A-(IY+d)	FD	96	d		|||V1|
SBC A,(HL)	A <-- A-(HL)-CY		9E			|||V1|
SBC A,(IX+d)	A <-- A-(IX+d)-CY	DD	9E	d		|||V1|
SBC A,(IY+d)	A <-- A-(IY+d)-CY	FD	9E	d		|||V1|
CP r	A-r		r | A B C D E H L | BF B8 B9 BA BB BC BD			|||V1|
CP n	A-n		FE	n		|||V1|

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNC V
CP (HL)	A-(HL)		BE			|||V1|
CP (IX+d)	A-(IX+d)	DD	BE	d		|||V1|
CP (IY+d)	A-(IY+d)	FD	BE	d		|||V1|
INC r	r <-- r+1		r: A B C D E H L 3C Ø4 ØC 14 1C 24 2C			|||VØ.
INC (HL)	(HL) <-- (HL)+1		34			|||VØ.
INC (IX+d)	(IX+d)<-- (IX+d)+1	DD	34	d		|||VØ.
INC (IY+d)	(IY+d)<-- (IY+d)+1	FD	34	d		|||VØ.
DEC r	r <-- r-1		r: A B C D E H L 3D Ø5 ØD 15 1D 25 2D			|||V1.
DEC (HL)	(HL) <-- (HL)-1		35			|||V1.
DEC (IX+d)	(IX+d)<-- (IX+d)-1	DD	35	d		|||V1.
DEC (IY+d)	(IY+d)<-- (IY+d)-1	FD	35	d		|||V1.

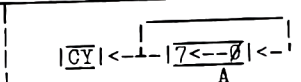
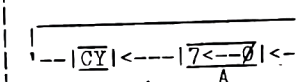
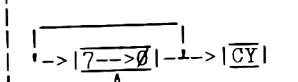
Mikrobefehlsliste - Z8Ø Operationscodes — Blatt 12

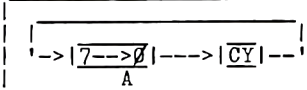
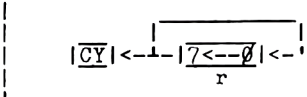
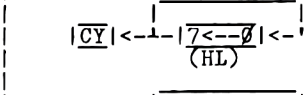
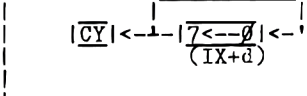
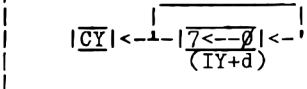
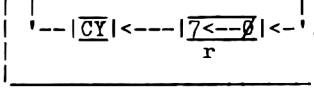
Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
Allgemeine Arithmetik -und Steuerbefehle						
DAA	Dezimalkorrektur nach Operation mit gepackten Zahlen		27			|||P.|
CPL	A <-- $\bar{A}$		2F			..1.1.
NEG	A <-- Ā+1	ED	44			|||V1|
CCF	CY <-- $\overline{CY}$		3F			..X.Ø|
SCF	CY <-- 1		37			..Ø.Ø1
NOP	keine Operation		ØØ			.....
HALT	CPU im Halt-Zustand		76			.....
DI	IFF <-- Ø		F3			.....
EI	IFF <-- 1		FB			.....
IM Ø	Setzen des Interrupt-Mode Ø	ED	86			.....

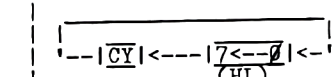
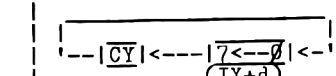
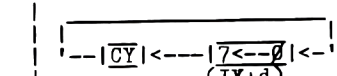
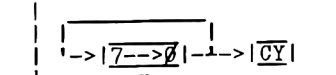
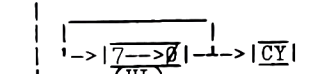
Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
IM 1	Setzen des Interrupt-Mode 1	ED	96			.....
IM 2	Setzen des Interrupt-Mode 2	ED	9E			.....
8-Bit-Logik-Befehle						
A | r | & | v | Ø Ø | Ø | Ø | Ø | Ø Ø | 1 | Ø | 1 | 1 1 | Ø | Ø | 1 | 1 1 | 1 | 1 | 1 | Ø						
AND r	A <-- A&r		r: A B C D E H L A7 AØ A1 A2 A3 A4 A5			||1PØØ
AND n	A <-- A&n		E6	n		||1PØØ
AND (HL)	A <-- A&(HL)		A6			||1PØØ
AND (IX+d)	A <-- A&(IX+d)	DD	A6	d		||1PØØ
AND (IY+d)	A <-- A&(IY+d)	FD	A6	d		||1PØØ

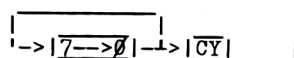
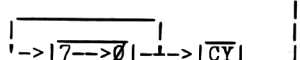
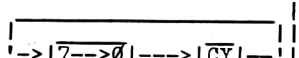
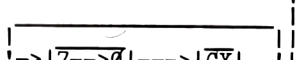
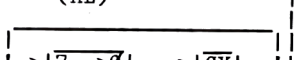
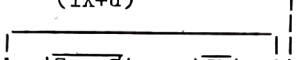
Befehl	Operation		Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNC V
OR r	A	<-- Avr		r | A B C D E H L | B7 BØ B1 B2 B3 B4 B5			||ØPØØ
OR n	A	<-- Avn		F6	n		||ØPØØ
OR (HL)	A	<-- Av(HL)		B6			||ØPØØ
OR (IX+d)	A	<-- Av(IX+d)	DD	B6	d		||ØPØØ
OR (IY+d)	A	<-- Av(IY+d)	FD	B6	d		||ØPØØ
XOR r	A	<-- A⊕r		r | A B C D E H L | AF A8 A9 AA AB AC AD			||ØPØØ
XOR n	A	<-- A⊕n		EE	n		||ØPØØ
XOR (HL)	A	<-- A⊕(HL)		AE			||ØPØØ
XOR (IX+d)	A	<-- A⊕(IX+d)	DD	AE	d		||ØPØØ
XOR (IY+d)	A	<-- A⊕(IY+d)	FD	AE	d		||ØPØØ

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
16-Bit-Arithmetik-Befehle						
ADD HL,dd	HL <-- HL+dd		dd | BC DE HL SP Ø9 19 29 39			..X.Ø|
ADD IX,dd	IX <-- IX+dd	DD	dd | BC DE IX SP Ø9 19 29 39			..X.Ø|
ADD IY,dd	IY <-- IY+dd	FD	dd | BC DE IY SP Ø9 19 29 39			..X.Ø|
ADC HL,dd	HL <-- HL+dd+CY	ED	dd | BC DE HL SP 4A 5A 6A 7A			||XVØ|
SBC HL,dd	HL <-- HL-dd-CY	ED	dd | BC DE HL SP 42 52 62 72			||XVØ|
INC dd	dd <-- dd+1		dd | BC DE HL SP Ø3 13 23 33			.....
INC IX	IX <-- IX+1	DD	23			.....

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNC V
INC IY	IY <-- IY+1	FD	23			.....
DEC dd	dd <-- dd-1		dd: BC ØB; DE 1B; HL 2B; SP 3B			.....
DEC IX	IX <-- IX-1	DD	2B			.....
DEC IY	IY <-- IY-1	FD	2B			.....
Rotations- und Verschiebebefehle						
RLCA			Ø7			..Ø.Ø.
RLA			17			..Ø.Ø.
RRCA			ØF			..Ø.Ø.

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNC V
RRA			1F			..Ø.Ø|
RLC r		CB	r | A B C D E H L ---|---| |Ø7 ØØ Ø1 Ø2 Ø3 Ø4 Ø5			||ØPØ|
RLC (HL)		CB	Ø6			||ØPØ|
RLC (IX+d)		DD CB d	Ø6			||ØPØ|
RLC (IY+d)		FD CB d	Ø6			||ØPØ|
RL r		CB	r | A B C D E H L ---|---| |17 1Ø 11 12 13 14 15			||ØPØ|

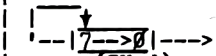
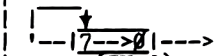
Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNC V
RL (HL)		CB	16			‖ØPØ‖
RL (IX+d)		DD CB d	16			‖ØPØ‖
RL (IY+d)		FD CB d	16			‖ØPØ‖
RRC r		CB	r | A B C D E H L ---- | ØF Ø8 Ø9 ØA ØB ØC ØD			‖ØPØ‖
RRC (HL)		CD	ØE			‖ØPØ‖

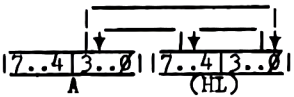
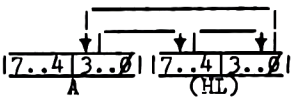
Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNC V
RRC (IX+d)		DD CB d	ØE			||ØPØ|
RRC (IY+d)		FD CB d	ØE			||ØPØ|
RR r		CB	r | A B C D E H L | 1F 18 19 1A 1B 1C 1D			||ØPØ|
RR (HL)		CB	1E			||ØPØ|
RR (IX+d)		DD CB d	1E			||ØPØ|
RR (IY+d)		FD CB d	1E			||ØPØ|

Mikrobefehlsliste - Z8Ø Operationscodes

Blatt 20

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
SLA r	|$\overline{CY}$|<---|$\overline{7<--Ø}$|<-Ø r	CB	r | A B C D E H L |27 2Ø 21 22 23 24 25			||ØPØ|
SLA (HL)	|$\overline{CY}$|<---|$\overline{7<--Ø}$|<-Ø (HL)	CB	26			||ØPØ|
SLA (IX+d)	|$\overline{CY}$|<---|$\overline{7<--Ø}$|<-Ø (IX+d)	DD CB d	26			||ØPØ|
SLA (IY+d)	|$\overline{CY}$|<---|$\overline{7<--Ø}$|<-Ø (IY+d)	FD CB d	26			||ØPØ|
SRA r	|$\overline{}$| |--|$\overline{7-->Ø}$|--->|$\overline{CY}$| r	CB	r | A B C D E H L |2F 28 29 2A 2B 2C 2D			||ØPØ|
SRA (HL)	|$\overline{}$| |--|$\overline{7-->Ø}$|--->|$\overline{CY}$| (HL)	CB	2E			||ØPØ|

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
SRA (IX+d)		DD CB	 2E			||ØPØ|
SRA (IY+d)		FD CB d	 2E			||ØPØ|
SRL r	Ø-> |7-->Ø| ---> |CY| (r)	CB	r: A B C D E H L 3F 38 39 3A 3B 3C 3D			||ØPØ|
SRL (HL)	Ø-> |7-->Ø| ----> |CY| (HL)	CB	3E			||ØPØ|
SRL (IX+d)	Ø-> |7-->Ø| ---> |CY| (IX+d)	DD CB d	 3E			||ØPØ|
SRL (IY+d)	Ø-> |7-->Ø| ---> |CY| (IY+d)	FD CB d	 3E			||ØPØ|

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNC V
RLD		ED	6F			||ØPØ.
RRD		ED	67			||ØPØ.
BIT b,r	Z <-- $\bar{r}_b$	CB	b\r | A B C D E H L Ø |47 4Ø 41 42 43 44 45 1 |4F 48 49 4A 4B 4C 4D 2 |57 5Ø 51 52 53 54 55 3 |5F 58 59 5A 5B 5C 5D 4 |67 6Ø 61 62 63 64 65 5 |6F 68 69 6A 6B 6C 6D 6 |77 7Ø 71 72 73 74 75 7 |7F 78 79 7A 7B 7C 7D			X|1XØ.
BIT b,(HL)	Z <-- $\overline{(HL)}_b$	CB	b | Ø 1 2 3 4 5 6 7 |46 4E 56 5E 66 6E 76 7E			X|1XØ.

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNC V
BIT b,(IX+d)	Z <-- $\overline{(IX+d)_b}$	DD CB d	b | Ø 1 2 3 4 5 6 7 | 46 4E 56 5E 66 6E 76 7E			X|1XØ.
BIT b,(IY+d)	Z <-- $\overline{(IY+d)_b}$	FD CB d	b | Ø 1 2 3 4 5 6 7 | 46 4E 56 5E 66 6E 76 7E			X|1XØ.
SET b,r	r_b <-- 1		b\r | A B C D E H L			.....
		CB	Ø | C7 CØ C1 C2 C3 C4 C5 1 | CF C8 C9 CA CB CC CD 2 | D7 DØ D1 D2 D3 D4 D5 3 | DF D8 D9 DA DB DC DD 4 | E7 EØ E1 E2 E3 E4 E5 5 | EF E8 E9 EA EB EC ED 6 | F7 FØ F1 F2 F3 F4 F5 7 | FF F8 F9 FA FB FC FD			
SET b,(HL)	$(HL)_b$ <-- 1		b | Ø 1 2 3 4 5 6 7			.....
		CB	| C6 CE D6 DE E6 EE F6 FE			

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
SET b,(IX+d)	$(IX+d)_b$ <-- 1	DD CB d	b: Ø 1 2 3 4 5 6 7 C6 CE D6 DE E6 EE F6 FE			.....
SET b,(IY+d)	$(IY+d)_b$ <-- 1	FD CB d	b: Ø 1 2 3 4 5 6 7 C6 CE D6 DE E6 EE F6 FE			.....
RES b,r	r_b <-- Ø	CB	b\r: A B C D E H L Ø: 87 80 81 82 83 84 85 1: 8F 88 89 8A 8B 8C 8D 2: 97 90 91 92 93 94 95 3: 9F 98 99 9A 9B 9C 9D 4: A7 A0 A1 A2 A3 A4 A5 5: AF A8 A9 AA AB AC AD 6: B7 B0 B1 B2 B3 B4 B5 7: BF B8 B9 BA BB BC BD			.....
RES b,(HL)	$(HL)_b$ <-- Ø	CB	b: Ø 1 2 3 4 5 6 7 86 8E 96 9E A6 AE B6 BE			.....
RES b,(IX+d)	$(IX+d)_b$ <-- Ø	DD CB d	b: Ø 1 2 3 4 5 6 7 86 8E 96 9E A6 AE B6 BE			.....

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
RES b,(IY+d)	$(IY+d)_b$ <-- Ø	FD CB d	b: Ø 1 2 3 4 5 6 7 86 8E 96 9E A6 AE B6 BE			.....
Sprungbefehle						
JP nn	PC <-- nn		C3	n_L	n_H	.....
JP cc,nn	PC <-- nn		cc: C2	n_L	n_H	.....
	Z = Ø		NZ: C2	n_L	n_H	.....
	Z = 1		Z: CA	n_L	n_H	.....
	C = Ø		NC: D2	n_L	n_H	.....
	C = 1		C: DA	n_L	n_H	.....
	P = Ø		PO: E2	n_L	n_H	.....
	P = 1		PE: EA	n_L	n_H	.....
	S = Ø		P: F2	n_L	n_H	.....
	S = 1		M: FA	n_L	n_H	.....
JP (HL)	PC <-- HL		E9			.....
JP (IX)	PC <-- IX	DD	E9			.....
JP (IY)	PC <-- IY	FD	E9			.....

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
JR e	PC <-- PC+e		18	e-2		.....
JR cc,e	PC <-- PC+e		cc | ---- | ----			
	Z = Ø		NZ | 2Ø			.....
	Z = 1		Z | 28			.....
	C = Ø		NC | 3Ø			.....
	C = 1		C | 38			.....
DJNZ e	PC <-- PC+e, wenn B ≠ Ø		1Ø	e-2		.....
	B <-- B-1					
CALL nn	(SP-1)<-- PC_H		CD	n_L	n_H	.....
	(SP-2)<-- PC_L					
	PC <-- nn					
	SP <-- SP-2					
CALL cc,nn	(SP-1)<-- PC_H					
	(SP-2)<-- PC_L					

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
	PC <-- nn					
	SP <-- SP-2					
	Z = Ø		NZ C4	n_L	n_H	.....
	Z = 1		Z CC	n_L	n_H	.....
	C = Ø		NC D4	n_L	n_H	.....
	C = 1		C DC	n_L	n_H	.....
	P = Ø		PO E4	n_L	n_H	.....
	P = 1		PE EC	n_L	n_H	.....
	S = Ø		P F4	n_L	n_H	.....
	S = 1		M FC	n_L	n_H	.....
RET cc	PC_L <-- (SP)					
	PC_H <-- (SP+1)					
	SP <-- SP+2					
	Z = Ø		NZ CØ			.....
	Z = 1		Z C8			.....
	C = Ø		NC DØ			.....
	C = 1		C D8			.....
	P = Ø		PO EØ			.....
	P = 1		PE E8			.....
	S = Ø		P FØ			.....
	S = 1		M F8			.....

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNC V
RET	PC_L <-- (SP) PC_H <-- (SP+1) SP <-- SP+2		C9			.....
RST p	(SP-1)<-- PC_H (SP-2)<-- PC_L PC_H <-- Ø PC_L <-- p SP <-- SP-2		p: Ø 1 2 3 4 5 6 7 C7 CF D7 DF E7 EF F7 FF			
Ein -und Ausgabebefehle						
IN A,(n)	A <-- n		DB	n		.....
IN r,(C)	r <-- (C)	ED	r: A B C D E H L 78 4Ø 48 5Ø 58 6Ø 68			.|P|Ø|

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
INI	B <-- B-1 HL <-- HL+1 (HL) <-- (C) B <-- B-1	ED	A2			.|XX1X c
INIR	(HL) <-- (C) B <-- B-1 HL <-- HL+1 wdh. bis B = Ø	ED	B2			.1XX1X
IND	(HL) <-- (C) B <-- B-1 HL <-- HL-1	ED	AA			.|XX1X c
INDR	(HL) <-- (C) B <-- B-1 HL <-- HL-1 wdh. bis B = Ø	ED	BA			.1XX1X
OUT (n),A	n <-- A		D3	n		.....
OUT C,r	(C) <-- r	ED	r | A B C D E H L | 79 41 49 51 59 61 69			.....

Befehl	Operation	Vor-byte	1.Byte	2.Byte	3.Byte	Flags SZHPNC V
OUTI	(C) <-- (HL) B <-- B-1 HL <-- HL+1	ED	A3			.|XX1X c
OTIR	(C) <-- (HL) B <-- B-1 HL <-- HL+1 wdh. bis B = Ø	ED	B3			.1XX1X
OTDR	(C) <-- (HL) B <-- B-1 HL <-- HL-1 wdh. bis B = Ø	ED	BB			.1XX1X
OUTD	(C) <-- (HL) B <-- B-1 HL <-- HL-1	ED	AB			.|XX1X c

Zeichenerklaerung:

a	P-Flag ist Ø, wenn das Ergebnis von BC-1 = Ø; sonst P = 1
b	Z-Flag ist 1, wenn A = (HL); sonst Z = Ø
IFF	Interrupt - Annahme - Flip - Flop
c	Wenn das Ergebnis B-1 = Ø, dann wird das Z-Flag gesetzt, sonst ist es geloescht.
.	Flag wird nicht beeinflusst
X	Flag unbestimmt
|	Flag entsprechend dem Ergebnis der Operation gesetzt/rueckgesetzt
Ø	Flag rueckgesetzt
1	Flag gesetzt
P	gerade Paritaet
V	Ueberlauf
CY	Carry - Flag
n_L	niederwertiges Byte der Adresse nn
n_H	hoeherwertiges Byte der Adresse nn
IX_L (IY_L, PC_L, dd_L)	niederwertiger Teil des Doppelregisters
IX_H (IY_H, PC_H, dd_H)	hoeherwertiger Teil des Doppelregisters

9.10. Mikrobefehlsliste - 8080-Operationscodes

Mikrobefehlsliste - 8080 Operationscodes — Blatt 1

Befehl	Operation	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
8-bit-Ladebefehle					
MOV r,r'	r <-- r'	r/r' | A B C D E H L A | 7F 78 79 7A 7B 7C 7D B | 47 40 41 42 43 44 45 C | 4F 48 49 4A 4B 4C 4D D | 57 50 51 52 53 54 55 E | 5F 58 59 5A 5B 5C 5D H | 67 60 61 62 63 64 65 L | 6F 68 69 6A 6B 6C 6D			.....
MVI r,n	r <-- n	r | A B C D E H L | 3E 06 0E 16 1E 26 2E	n		.....
MOV r,M	r <-- (HL)	r | A B C D E H L | 7E 46 4E 56 5E 66 6E			.....
MOV M,r	(HL) <-- r	r | A B C D E H L | 77 70 71 72 73 74 75			.....
MVI M,n	(HL) <-- n	36	n		.....

Befehl	Operation	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
LDAX d	A <-- (BC) A <-- (DE)	d: B. D ØA 1A			.....
STAX d	(BC) <-- A (DE) <-- A	d: B D Ø2 12			.....
LDA nn	A <-- (nn)	3A	n_L	n_H	.....
STA nn	(nn) <-- A	32	n_L	n_H	.....
16-bit-Ladebefehle					
LXI d,nn	BC <-- nn DE <-- nn HL <-- nn SP <-- nn	d: BC DE HL SP Ø1 11 21 31	n_L	n_H	.....
LHLD nn	H <-- (nn+1) L <-- (nn)	2A	n_L	n_H	.....
SHLD nn	(nn+1) <-- H (nn) <-- L	22	n_L	n_H	.....
SPHL	SP <-- HL	F9			.....

Mikrobefehlsliste - 8Ø8Ø Operationscodes Blatt 3

Befehl	Operation	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
Indirekte Registeroperationen					
PUSH PSW	(SP-2)<-- F (SP-1)<-- A	F5			.....
PUSH d	(SP-2)<-- d_L (SP-1)<-- d_H	d | BC DE HL ---- | ---------- | C5 D5 E5			.....
POP PSW	A <-- (SP+1) F <-- (SP)	F1			.....
POP d	d_H <-- (SP+1) d_L <-- (SP)	d | BC DE HL ---- | ---------- | C1 D1 E1			.....
Austauschbefehle					
XCHG	DE <--> HL	EB			.....
XTHL	H <--> (SP+1) L <--> (SP)	E3			.....

Mikrobefehlsliste - 8080 Operationscodes — Blatt 4

Befehl	Operation	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
8-bit-Arithmetik					
ADD r	A <-- A+r	r: A B C D E H L 87 80 81 82 83 84 85			|||V0|
ADD M	A <-- A+(HL)	86			|||V0|
ADI n	A <-- A+n	C6	n		|||V0|
ADC r	A <-- A+r+CY	r: A B C D E H L 8F 88 89 8A 8B 8C 8D			|||V0|
ADC M	A <-- A+(HL)+CY	8E			|||V0|
ACI n	A <-- A+n+CY	CE	n		|||V0|
SUB r	A <-- A-r	r: A B C D E H L 97 90 91 92 93 94 95			|||V1|
SUB M	A <-- A-(HL)	96			|||V1|
SUI n	A <-- A-n	D6	n		|||V1|

Mikrobefehlsliste - 8Ø8Ø Operationscodes Blatt 5

Befehl	Operation	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
SBB r	A <-- A-r-CY	r: A B C D E H L 9F 98 99 9A 9B 9C 9D			|||V1|
SBB M	A <-- A-(HL)-CY	9E			|||V1|
SBI n	A <-- A-n-CY	DE			
CMP r	A-r	r: A B C D E H L BF B8 B9 BA BB BC BD			|||V1|
CMP M	A-(HL)	BE			|||V1|
CPI n	A-n	FE			|||V1|
INR r	r <-- r+1	r: A B C D E H L 3C Ø4 ØC 14 1C 24 2C			|||VØ.
INR M	(HL) <-- (HL)+1	34			|||VØ.

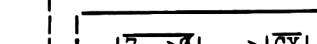
Befehl	Operation	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
DCR r	r <-- r-1	r: A B C D E H L 3D Ø5 ØD 15 1D 25 2D			|||V1.
DCR M	(HL) <-- (HL)-1	35			|||V1.
Allgemeine Arithmetik- und Steuerbefehle					
DAA	Dezimalkorrektur nach Operation mit gepackten Zahlen	27			|||P.
CMA	A <-- $\overline{A}$	2F			..1.1.
CMC	CY <-- $\overline{CY}$	3F			..X.Ø|
STC	CY <-- 1	37			..Ø.Ø1
NOP	keine Operation	ØØ			.....
HLT	CPU im Haltzustand	76			.....
DI	IFF <-- Ø	F3			.....
EI	IFF <-- 1	FB			.....

Mikrobefehlsliste - 8Ø8Ø Operationscodes Blatt 7

Befehl	Operation	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
8-bit-Logikbefehle					
ANA r	A <-- A&r	r: A B C D E H L A7 AØ A1 A2 A3 A4 A5			||1PØØ1
ANA M	A <-- A&(HL)	A6			||1PØØ
ANI n	A <-- A&n	E6	n		||1PØØ
ORA r	A <-- Avr	r: A B C D E H L B7 BØ B1 B2 B3 B4 B5			||ØPØØ
ORA M	A <-- Av(HL)	B6			||ØPØØ
ORI n	A <-- Avn	F6	n		||ØPØØ
XRA r	A <-- A@r	r: A B C D E H L AF A8 A9 AA AB AC AD			||ØPØØ
XRA M	A <-- A@(HL)	AE			||ØPØØ
XRI n	A <-- A@n	EE	n		||ØPØØ

Mikrobefehlsliste – 8Ø8Ø Operationscodes — Blatt 8

Befehl	Operation	1.Byte	2.Byte	3.Byte	Flags SZHPNC V
16-bit-Arithmetik					
DAD d	HL <-- HL+d	d: BC ØØ, DE 1Ø, HL 2Ø, SP 3Ø			..X.Ø|
INX d	d <-- d+1	d: BC Ø3, DE 13, HL 23, SP 33			.....
DCX d	d <-- d-1	d: BC ØB, DE 1B, HL 2B, SP 3B			.....
Verschiebebefehle					
RLC	|CY|<-|7<--Ø|<-A	Ø7			..Ø.Ø|
RRC	->|7-->Ø|->|CY| A	ØF			..Ø.Ø|
RAL	--|CY<--7|<---|Ø|<- A	17			..Ø.Ø|

Befehl	Operation	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
RAR		1F			..Ø.Ø.
Sprungbefehle					
JMP nn	PC <-- nn	C3	n_L	n_H	.....
Jcc nn		cc'			
	Z = Ø	NZ C2	n_L	n_H	.....
	Z = 1	Z CA	n_L	n_H	.....
	C = Ø	NC D2	n_L	n_H	.....
	C = 1	C DA	n_L	n_H	.....
	P = Ø	PO E2	n_L	n_H	.....
	P = 1	PE EA	n_L	n_H	.....
	S = Ø	P F2	n_L	n_H	.....
	S = 1	M FA	n_L	n_H	.....
PCHL	PC <-- HL	E9			.....

Befehl	Operation	1.Byte	2.Byte	3.Byte	Flags ------ SZHPNC V
CALL nn	(SP-1)<-- PC_H (SP-2)<-- PC_L PC <-- nn SP <-- SP-2	CD	n_L	n_H	.....
Ccc nn	(SP-1)<-- PC_H (SP-2)<-- PC_L PC <-- nn SP <-- SP-2				
		cc | BC			
	Z = 0	NZ | C4	n_L	n_H	.....
	Z = 1	Z | CC	n_L	n_H	.....
	C = 0	NC | D4	n_L	n_H	.....
	C = 1	C | DC	n_L	n_H	.....
	P = 0	PO | E4	n_L	n_H	.....
	P = 1	PE | EC	n_L	n_H	.....
	S = 0	P | F4	n_L	n_H	.....
	S = 1	M | FC	n_L	n_H	.....

Befehl	Operation	1.Byte	2.Byte	3.Byte	Flags SZHPNCV
RST p	(SP-1)<-- PC_H (SP-2)<-- PC_L PC_H <-- Ø PC_L <-- p SP <-- SP-2	p | Ø 1 2 3 4 5 6 7 | C7 CF D7 DF E7 EF F7 FF			
Ein-und Ausgabebefehle					
IN n	A <-- n	DB	n		.....
OUT n	n <-- A	D3	n		.....

Zeichenerklaerung:

X	Flag unbestimmt
|	Flag entsprechend dem Ergebnis der Operation gesetzt/rueckgesetzt
ø	Flag rueckgesetzt
1	Flag gesetzt
P	gerade Paritaet
V	Ueberlauf
CY	Carry - Flag
.	Flag wird nicht veraendert

n_L	niederwertiges Byte der Adresse nn
n_H	hoeherwertiges Byte der Adresse nn
d_L (PC_L)	niederwertiger Teil des Doppelregisters
d_H (PC_H)	hoeherwertiger Teil des Doppelregisters

III-12-12 Kv 1976/85